Lehrstuhl für Mensch-Maschine-Kommunikation
Technische Universität München

Hierarchical Language Modeling for One-Stage Stochastic Interpretation of Natural Speech

Matthias Thomae

Vollständiger Abdruck der
von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München
zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs
genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. Klaus Diepold

Prüfer der Dissertation:

1. apl. Prof. Dr.-Ing., Dr.-Ing. habil. Günther Ruske
2. Univ.-Prof. Dr. phil. nat. Helmut Seidl

Die Dissertation wurde am 21.11.2005 bei der
Technischen Universität München eingereicht und durch die
Fakultät für Elektrotechnik und Informationstechnik
am 15.03.2006 angenommen.

Danksagung

An erster Stelle möchte ich meinem Doktorvater, Prof. Günther Ruske, für das Vertrauen und die Unterstützung danken, die er mir während meiner Zeiten am Lehrstuhl für Mensch-Maschine-Kommunikation geschenkt hat, indem er mich als Werkstudent, Diplomand und Promovend unter seine Fittiche genommen und mich für die automatische Sprachverarbeitung begeistert hat. Diese Arbeit verdanke ich auch meinen Kollegen Robert Lieb und Tibor Fabian, die mit Prof. Ruske und mir im ODINS-Team Ihre Ideen und Ergebnisse geteilt haben, um gemeinsam die 1-stufige Sprachinterpretation zu beleuchten.

Ein herzliches Dankeschön geht auch an Prof. Diepold für den Prüfungsvorsitz, an Prof. Seidl für das Zweitgutachten zur Dissertation und an Prof. Rigoll für seine Unterstützung.

Für Korrekturvorschläge möchte ich Dr. Elmar Nöth danken, und Renate und Waldemar Goronzy dafür, dass sie mir das passende Umfeld für die "Initialzündung" zur schriftlichen Ausarbeitung geschaffen haben. Ganz besonderer Dank gilt meiner Verlobten, Dr. Silke Goronzy, die mir in allen Phasen der Promotion mit Rat und Tat beigestanden hat.

Ermöglicht haben diese Arbeit letztendlich meine Eltern, Elisabeth und Heinz, indem sie mich auf meinem Lebensweg in jeglicher Hinsicht gefördert und unterstützt haben, und immer für mich da waren.

Matthias Thomae
Erlangen, Juni 2006

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Speech is one of the most important means of communication between humans. Its main purpose is to exchange ideas and intentions between the communication partners, but it also transports various other kinds of information such as feelings or evidence about gender and age of the speaker. Most of the information about the speaker's ideas and intentions is carried by the uttered words and phrases.

Engineers denote the task of transcribing spoken utterances into sequences of words as automatic speech recognition. Most human beings are capable of performing this task effortlessly and almost perfectly, i.e. without making many errors. Furthermore, humans are able to recognize speech in many different and sometimes adverse situations, e.g. in traffic noise, in a busy restaurant, on a mobile phone with bad reception or when the communication partner's nose is blocked. In addition to acoustic corruptions, humans are able to handle other speech degradations such as accents or non-native speech, grammatically or semantically wrong utterances, or incomplete ones. Due to such difficulties, the performance of nowadays' speech recognition systems is far from being close to humans, despite the efforts and the progress that research into this field has made during previous decades.

The prospect of automatic systems with human speech recognition performance poses a challenge to many scientists, engineers and other enthusiasts. With speech being one of the most natural and efficient ways of human communication, speech technology has the potential to revolutionize the communication between humans and machines. However, in order to practically realize advanced human-machine communication scenarios like the dialogues between the crew of Starship Enterprise and the Ship's computer in Star Trek, the machine needs to understand the meaning of the spoken utterances. The *speech understanding* problem is the fundamental topic of this work. In order to recognize meaning, the machine is not necessarily required to produce a perfect word transcription. For the execution of an action, such as programming the food replicator or the turbolift, it must only correctly recognize the crew member's intention including important key words such as 'coffee' or 'bridge', but not exactly how users express their wishes.

The study of the meaning of linguistic utterances is called semantics. The task

of generating semantic representations from spoken utterances is called speech understanding. This task requires knowledge from two traditionally separate fields of research, namely *automatic speech recognition* (ASR) and *natural language understanding* (NLU). The latter is the process of converting textual utterances to meaning representations. This process is usually based on explicit morphological and syntactic analyses of the input text, i.e. it relies on knowing the structure of the words and the structural relationships between words. Such a 'deep' semantic analysis, however, is difficult to perform if the input text is not well-formed, i.e. if it contains morphological and syntactic errors. This has to be regarded if the output of a speech recognizer is used, as these systems can usually not rely on well-formed speech input and additionally produce errors themselves.

Because of the erroneous speech recognizer output and the computational difficulty to perform speech recognition and deep analyses in appropriate time, state-of-the-art speech understanding is not a simple application of ASR and NLU techniques. Instead of the deep semantic analysis, a robust semantic analysis is performed. The robustness is achieved by performing no or only little explicit syntactic and morphological analyses, and limiting the scope of speech understanding applications to narrow domains. Therefore, practical systems today operate in specialized contexts, so-called target domains, such as weather information, flight booking or account enquiries. A number of such systems have been described, e.g. EVAR, LIMSI ARISE, MIT JUPITER, SRI ATIS, SUNDIAL or TRAINS/TRIPS [BDHS95, FAM96, GAB+98, LRGB99, Pec93, ZSG+00]. The methods for speech understanding presented in this thesis are also based on the limited-domain assumption. Investigations are performed by use of a test system built from a speech data collection for a German airport information dialogue system scenario.

Speech understanding is traditionally carried out by combining ASR and NLU processing in a sequential manner. The tasks of mapping speech to text and text to meaning representations are performed by two separate processing stages. We denote this as a two-stage decoding process. This method has practical advantages: The decoders for both stages can theoretically be replaced by implementations which carry out the same task, enabling comparison and more flexibility for speech understanding system manufacturers. Moreover, the decoders can be developed independently of each other, and each can independently utilize methodologies and algorithms specialized for its task.

A tighter integration of ASR and NLU techniques can be achieved by **one-stage decoding**. One-stage decoders directly map speech to meaning representations. Hence, words do not necessarily play the central role, but can be treated as a processing unit among others. One-stage decoding approaches have the fundamental advantage that they obey the well-known principle of applying all available sources of knowledge as early as possible. This is not the case in two-stage speech understanding systems, where the speech recognizer makes decisions without considering semantic knowledge from the second stage. Yet, such early decisions can cause speech understanding errors which are avoided by one-stage decoding.

One-stage decoding may also be advantageous with regard to runtime behavior. If decoding is performed time-synchronously, the speech understanding result can be delivered with minimal delay. Yet, a conclusive comparison of runtime performance depends on a great variety of factors, and is not goal of this work. The one-stage approach also alleviates a practical problem: Inconsistent knowledge sources, such as diverging word vocabularies, are ultimately discovered during the generation of the integrated knowledge model. In two-stage systems, inconsistencies may remain unnoticed and cause problems during runtime.

Stahl et al. [SML97] describe a one-stage semantic decoder for natural speech input, realized by stochastic chart-parsing of extended context-free grammars. For an application to control a graphical editor, their system achieves semantic accuracies of 88.4% with the one-stage approach, and 86.0% with a two-stage decoder whose NLU stage only uses the best word sequence as input. Johnsen et al. [JHHS00] use Discrete Hidden-Markov Models for semantic processing in a Norwegian bus travel information system. Their semantic model is integrated into a speech recognizer instead of a conventional word-based language model. Acero et al. [AW04] propose a so-called semantically structured language model, which combines n-grams and context-free grammar rules in a single, stochastic language model. Like Johnsen, they use their semantic model in a regular speech recognizer instead of a conventional language model. In comparison to a two-stage decoder, whose semantic stage is fed with the best recognition result only, their corresponding one-pass system reduces topic classification errors substantially (from 6.8% to 3.8%) and slot identification errors slightly (from 9.0% to 8.8%).

This thesis also follows the basic one-stage decoding principle. In contrast to other publications, however, we also report how the two-stage speech understanding approach performs when its NLU stage operates with multiple alternative recognizer hypotheses instead of only the best one. Two-stage systems sometimes consider alternative word or utterance hypotheses to circumvent the problems caused by early decisions. In practice, the possible number of alternatives is limited by the computational resources available for the second stage. In this work, we specifically investigate how many alternatives are required to achieve a similar semantic performance as the one-stage system, and how many errors can be avoided if no or only few alternatives are available.

One-stage decoding methods for speech understanding demand simultaneous processing of knowledge from different fields such as acoustics, phonetics, (lexical) phonology, syntax and semantics in a single procedure. Each of these processing levels has its own special requirements regarding optimum modeling and decoding of knowledge. Hence, representation and processing of all this knowledge can be approached in two different ways: Either, models of different types are created, and a specially tailored, hybrid processing algorithm is devised for them. The aforementioned approaches by Stahl, Johnsen and Acero fall into this category, for example. Alternatively, the knowledge sources can be transfered to a **uniform knowledge representation**, so that a more generic processing algorithm can be utilized. This

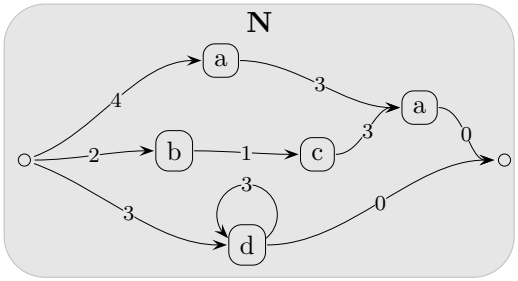


Figure 1.1: *Weighted transition network and equivalent weighted context-free rewrite rule.*

method was adopted in this thesis, because it allows flexible integration of knowledge without having to modify the decoder, as long as this knowledge is representable within the uniform modeling framework. This renders the approach more suitable for research purposes at least. Furthermore, decoder complexity is potentially lower, as only one algorithmic framework needs to be considered, instead of many.

The choice of a suitable uniform modeling framework is crucial. On the one hand its complexity must be low enough to enable (near-) realtime processing, on the other hand it needs sufficient expressive power to embed the different kinds of knowledge within it. For this thesis, a uniform, stochastic knowledge representation based on **weighted transition network hierarchies** (WTNH) was selected. In contrast to weighted finite-state transducers (WFST), which have been used as uniform framework for speech recognition (see e.g. [MPR02]), WTNH represent hierarchical information in an explicit form. Since WFST require implicit encoding of such hierarchical knowledge, tasks that require direct access to the model structure are complicated. Therefore and due to other reasons, WTNH are preferred in this work.

The basic building blocks of WTNH are weighted transition networks, which consist of labeled nodes and weighted transitions between these nodes. Weighted transition networks can also be viewed as weighted rewrite rules, so that WTNH represent rule sets or grammars, more precisely stochastic context-free grammars. Figure 1.1 depicts a simple example of a weighted transition network N and its equivalent representation as weighted context-free rewrite rule (see also Section 2.2.2). Assuming that the weights, which are attached to transitions between network nodes, are summed up along a path, this network e.g. produces the symbol sequences **aa** at a cost of 7, **bca** at a cost of 6, and **ddd** at a cost of 9. WTNH consist of a set of such transition networks and hierarchical dependencies between them, realized as references from nodes to subordinate networks.

From the viewpoint of natural language understanding, the expressiveness of this type of knowledge representation is comparably low. Due to this and in order to limit the scope of this thesis, comparably simple semantic knowledge is represented

by WTNH. Therefore, we substitute the term speech understanding with the less ambitious term **speech interpretation** for the task pursued here.

This thesis mainly focuses on the syntactic-semantic modeling levels of WTNH, denoted as **hierarchical language model** (HLM). As a special characteristic of HLM, they are created by combining different rule-based and data-driven language modeling techniques. From the mixture of these two fundamentally different approaches, we expect to limited the effort for model construction and to better cope with the lack of sufficient amounts of training data. The hierarchical structure of HLM is defined explicitly in this work, i.e. no automatic hierarchy-building techniques are applied. Furthermore, HLM are trained from fully annotated data. The data sparsity problem, which many statistical modeling approaches suffer from, is studied by means of different smoothing techniques.

In the context of limited-domain conversational dialogue or control applications, speech interpretation systems should ideally be able to cope with natural speech input. Therefore, HLM specifically need to be robust against natural speech phenomena and against unknown words. Both of these problems are specifically addressed in this work. A word which is not contained in the system vocabulary is called unknown word or *out-of-vocabulary* (OOV) word. OOV words are an important problem for practical speech recognition and interpretation systems, because they are silently misrecognized if no special arrangements for their treatment are performed. Especially in the context of limited domains, where the system vocabulary typically consists of merely a few hundred or thousand words, a high rate of OOV words can be expected. Therefore, techniques to build suitable OOV word models and to integrate them into HLM are examined in this thesis. Natural speech phenomena are e.g. pauses, laughter, coughing, hesitations, mumbling, cut-off words, self-corrections and ungrammatical utterances. Some of these phenomena are described by special models for filler words and non-speech sounds. Other are alleviated by smoothing of likelihood distributions, combination of data-driven and rule-based language modeling techniques and unknown word modeling.

The evaluation of the studied speech interpretation methods is also a fundamental topic in this work. The standard performance measure in speech recognition is word accuracy, which summarizes how many words are correctly recognized on average. Another common measure, which judges language models alone, is test-set perplexity. Both measures are adapted to the hierarchical speech interpretation task considered in this work. The notion of perplexity is applied to HLM. More importantly, a suitable 'end-to-end' performance measure, the so-called **semantic tree node accuracy**, is proposed. In contrast to standard approaches, the novel measure is computed by tree matching instead of sequence matching. It is therefore expected to be more flexible when assessing partially correct semantic tree representations. In order to evaluate the performance of unknown word models, receiver-operating-characteristics and figures-of-merit are computed.

This work is structured as follows: The uniform modeling approach for one-stage speech interpretation is presented in Chapter 2, starting with an overview over the basic processing stages and the fundamental differences to two-stage speech understanding systems. The basic one-stage decoding scheme for weighted transition network hierarchies is described in Chapter 3. The chapter is concluded with an experiment that quantifies the advantage of the one-stage decoding approach over a corresponding two-stage speech interpretation system. Chapter 4 discusses the evaluation methodology of this thesis, including semantic tree node accuracy. An experimental comparison between sequence matching and tree matching based evaluation measures is performed. This chapter also describes the airport information speech corpus utilized for model training and evaluation, and its annotation with semantic information.

The hierarchical language modeling approach is presented in Chapter 5. Rule-based and data-driven language modeling techniques and their use for HLM are discussed as well as modeling of various natural speech phenomena. Different experiments regarding smoothing techniques, the range of language model dependencies and distribution of likelihood values are illustrated and discussed. The chapter concludes with an experimental evaluation of the question whether semantic knowledge is able to improve word accuracy, since many scientists attribute importance to this measurement. Chapter 6 presents our unknown word modeling approach and its integration into HLM, along with various experimental results. In Chapter 7 the results and implications of this thesis are summarized, and motivation for future work is given.

Chapter 2

A Uniform Model for One-Stage Speech Interpretation

In this chapter, the basic uniform modeling framework for one-stage interpretation of natural speech is discussed. The initial section gives a brief overview over the basic building blocks of typical speech understanding systems and the fundamental differences to the one-stage speech interpretation approach of this thesis. Section 2.2 then reviews suitable approaches for uniform knowledge modeling, based on formal language theory. Specifically, regular language representation by regular expressions and finite-state automata, and context-free language representation by automata hierarchies are discussed. The selected modeling framework, namely weighted transition network hierarchies, and its similarities with stochastic context-free grammars is then discussed in Section 2.3. After introducing the basic structure, the notion of hierarchy levels within the uniform modeling framework is explained. Then, the result of utilizing this knowledge model for speech interpretation is considered, in the shape of semantic trees. In the final part of this chapter, an overview is given over how the different knowledge sources, namely Hidden-Markov Models (HMM), lexicon and semantic model, can be integrated into the chosen framework.

2.1 Overview: One-Stage vs. Two-Stage Decoding

This section provides an insight into the basic components and fundamental differences between classical two-stage speech understanding systems and the one-stage speech interpretation approach pursued in this work. Figure 2.1 depicts the basic building blocks of both types of systems, along with the knowledge fields which the blocks represent, namely acoustics, phonetics, lexical phonology, syntax and semantics. The first stage of a **two-stage speech understanding system** consists of a speech recognizer (see upper part of Figure 2.1). In most state-of-the-art continuous speech recognition systems, HMM are employed to describe fundamental acoustic-phonetic units of spoken language such as phonemes or syllables. Phonemes are often used in a context dependent version, e.g. as diphones or triphones, rather than

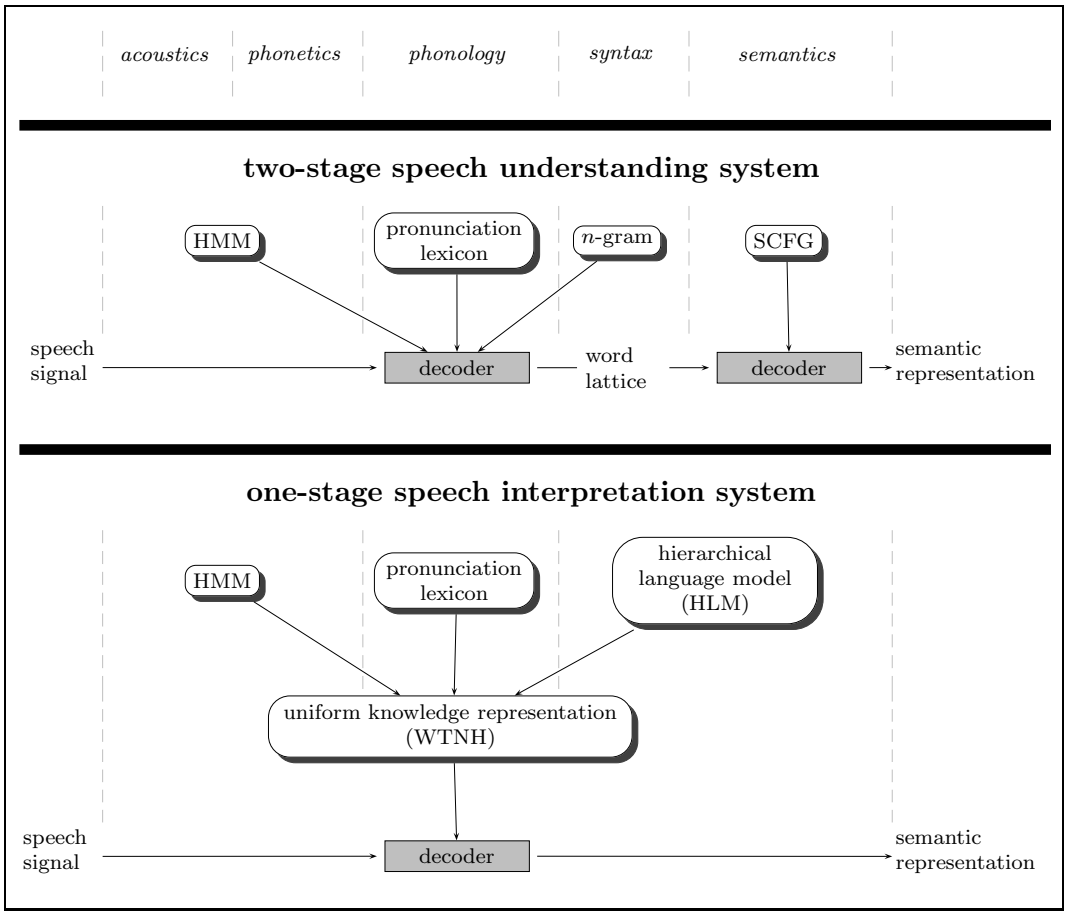


Figure 2.1: *Basic processing levels of a typical two-stage speech understanding system and our uniform one-stage speech interpretation system.*

context independent monophones. The mapping from phonemes to words is carried out by a pronunciation lexicon. There may be different non-standard pronunciations for a word, called pronunciation variants.

A speech recognizer captures the syntactic structure of spoken language by means of a so-called language model. An explicit syntactic analysis is usually not performed in speech recognition. Instead, the syntactic structure is described by statistical relations between words, derived from a textual corpus of training sentences. The mostly used statistical language models are n-gram language models, of which several variants with abilities to cope with training data sparsity exist. The speech recognizer outputs its best hypothesis of a spoken utterance as a sequence of words. Alternative hypotheses are usually produced on the utterance level (as n-best sentence lists) or on the word level (as word lattices or word graphs).

The second stage of a two-stage speech understanding system decodes a semantic representation from the textual representation of a spoken utterance. The semantic

(and possibly syntactic) knowledge for the second-stage decoder is typically modeled by a *stochastic context-free grammar* (SCFG). The decoding process is also called *parsing* in this context. A *parse tree* represents the semantic structure, where each node of the tree corresponds to a SCFG rule of the parse.

One fundamental difference between two-stage speech understanding and **one-stage speech interpretation** is that the latter utilizes only one model to describe the syntactic-semantic properties of the target domain. This modeling task is performed by the *hierarchical language model* (HLM) in this thesis. The HLM is a special syntactic-semantic language model which can be integrated together with HMM and pronunciation lexicon into a uniform modeling framework, more precisely into a weighted transition network hierarchy (WTNH). By use of this integrated knowledge model, the one-stage decoder directly translates spoken utterances into semantic representations, without explicitly creating an intermediate orthographic representation.

Please note that the basic preprocessing of the speech signal is not shown in Figure 2.1, although this task is also performed by the (first) decoder. Moreover, the uniform knowledge model does not explicitly contain the state probability density functions (PDF) of the HMM, but only their state transition structure and weighting. The PDF are kept in a separate data structure with an algorithm for computing HMM emission probabilities. An interface to the uniform modeling framework enables the transfer of computation results.

2.2 Uniform Knowledge Model Selection

In this section, we will discuss criteria for choosing an appropriate uniform knowledge model for one-stage speech interpretation. The uniform modeling approach is preferred for the one-stage decoding task of this thesis over hybrid modeling with a specially tailored processing algorithm such as in [GA00]. The main reasons for this choice are its extensibility with new types of knowledge models without requiring decoder modification, and a potentially less complex decoder. While the hybrid approach may lend itself to computationally more attractive decoder designs, this was not a focus of this thesis.

The flexibility of the uniform approach is limited by the expressive power of the underlying knowledge model. Simultaneously, the expressive power of a knowledge model determines the size of the search space, which is generally coupled with the computational requirements for decoding. Hence, the complexity of the uniform representation must be low enough to maintain computational tractability.

The process of interpreting speech, or generating meaning representations from spoken utterances, can be viewed as a symbol processing task, if not the speech signal itself, but preprocessed acoustic observations (feature vectors) are used as the basic symbols. The decoder's task then consists of finding the 'correct' symbolic meaning representation for a given sequence of acoustic observations. Because of the diversity of speech, language and meaning, the knowledge models for speech

interpretation are of statistical nature. Therefore, we redefine the task of the speech interpreter as to find the *most likely* meaning representation for a sequence of acoustic observations, given the available knowledge models. The symbols are usually not translated directly, but intermediate levels of representation are introduced to facilitate the design of appropriate models. Thus, the acoustic observation symbols are first translated into sub-word units, typically phonemes. Phonemes are mapped to word symbols. The words are further processed to yield semantic symbols.

2.2.1 Formal Languages and Grammars

In computational science, the standard means for symbol processing are formal languages. A formal language defines a possibly infinitely large set of finite-length sequences of symbols (strings) from a finite symbol alphabet. Formal languages are usually described by a set of production rules. The rule set together constitutes a formal grammar, more precisely a generative grammar. A production rule is generally expressed as

$$LHS \rightarrow RHS$$

where *LHS* denotes the *left-hand side* and *RHS* the *right-hand side* of the rule. A production rule, also called rewrite rule, declares that *LHS produces RHS* or *LHS is rewritten by RHS*. The *LHS* and *RHS* of production rules consist of terminal symbols from a finite alphabet Σ and non-terminal symbols from a finite alphabet N. Non-terminals are usually represented by uppercase letters, terminals by lowercase letters.

A formal grammar is able to 'generate' symbol sequences by subsequently rewriting parts of a symbol sequence. The rewriting is carried out by replacing the part of the symbol sequence corresponding to a *LHS* of a production rule by its *RHS*. The rules which can be applied at the beginning of the production process are called start rules. Their *LHS* consists of the special non-terminal symbol S, called start symbol. The special terminal symbol ϵ is called the empty symbol. A valid (final) symbol sequence of the language contains only terminal symbols.

Apart from producing symbol sequences, formal grammars can also be used reversely to analyze symbol sequences, i.e. to split up an input string into tokens. This process consists of determining which rules need to be 'fired' (applied) to produce the desired symbol sequence, and is called parsing. The result of parsing is a parse tree, whose nodes represent the fired rules, and whose structure corresponds to the order in which rules were fired. By looking for a successful parse, formal grammars are also able to decide whether an input string can be recognized at all, i.e. if it is a valid string of a language or not. This property is e.g. used for syntax checking.

A basic categorization of formal languages according to their expressive power was done by Noam Chomsky [Cho56, Cho59]. His Chomsky Hierarchy (see Table 2.1) defines four classes or types of formal grammars. From the most general grammar class (type 0) to the most restricted grammar class (type 3) these are called

type	formal grammar	formal language	equivalent machine
0	unrestricted	recursively enumerable	Turing machine
1	context-sensitive	context-sensitive	linear-bounded automaton
2	context-free	context-free	pushdown automaton
3	regular	regular	finite-state automaton

Table 2.1: *Chomsky Hierarchy [HU79].*

unrestricted grammars, *context-sensitive grammars*, *context-free grammars* and *regular grammars*. The formal languages that these classes of grammars can produce are called *recursively enumerable languages*, *context-sensitive languages*, *context-free languages* and *regular languages*, respectively. Each Chomsky type is associated with an abstract machine or automaton, which is capable of recognizing all languages of that type. The four types of languages form a strict hierarchy, i.e. a language of type t_l can always be described by a grammar of the same or a more expressive type $t \leq t_l$, whereas the opposite is at most true in special cases. For example, certain types of context-free grammars, namely right-linear or left-linear context-free grammars, are equivalent to regular grammars.

The grammar classes with most practical relevance are context-free grammars and regular grammars, because there are efficient parser implementations which render these grammars computationally tractable for a number of applications. These two kinds of grammars are also the basis for the modeling approach described in this section.

2.2.2 Regular Expressions and Finite-State Automata

Regular grammars consist of rules of the form

$$\begin{array}{lcl} A & \rightarrow & Ba \\ A & \rightarrow & a \end{array} \quad \text{or} \quad \begin{array}{lcl} A & \rightarrow & aB \\ A & \rightarrow & a \end{array} \tag{2.1}$$

In the case depicted at the left side of Equation 2.1 they are called *left regular grammars*, in the case at the right side of Equation 2.1 *right regular grammars*. A more familiar form of representing regular languages are *regular expressions*. A regular expression consists of a string of symbols and operators on these symbols. There are different sets of regular expression operators in use, Table 2.2 shows a listing of frequently used ones. Regular expressions are practically applied e.g. to define search patterns in text editors and many Unix tools, or for string manipulation in programming languages. A rule with a single non-terminal at its LHS and a

operator	notation	meaning
concatenation	$a\,b$	a followed by b
alternation	$a\|b$	a or b
option	$a?$	zero times or once
Kleene star	a^*	zero, one or several times
Kleene plus	a^+	one or several times
grouping	$(a\,b)^*$	scope and precedence

Table 2.2: *Common regular expression operators.*

regular expression at its RHS is called *context-free rewrite rule*:

$$A \rightarrow \textit{regexp}$$

Regular expressions can also be weighted, by assigning costs to symbols (see e.g. [Spr99b]). An example for such a weighted regular expression as part of a weighted context-free rewrite rule is shown in Figure 1.1 of Chapter 1.

Regular grammars and regular expressions have equivalent representations as **finite-state automata** (FSA), which are abstract machines with a finite amount of memory in the form of states. Please note that the following discussion on FSA is not meant as a complete description of finite-state theory, but rather as a brief outline of some facts important for this work. For a more detailed discussion on this topic the interested reader is referred to [Moh97].

A finite-state automaton A is defined as a tuple $A = (\Sigma, S, T, i, f)$ over an alphabet Σ, a finite set of states S, a finite set of transitions T, an initial state $i \in S$ and a final state $f \in S$. A transition $t \in T$ has a source state and a destination state and is depicted as an arc. Finite-state automata are also called *transition networks*, and the states are then denoted as nodes.

As representatives of formal languages, an essential component of FSA are the language symbols. There are two types of finite-state machines, namely *Mealy* and *Moore* machines, which differ in their placement of symbols. Mealy machines carry their symbols on the transitions, whereas Moore machines utilize the states as symbols carriers. Mealy and Moore machines can be converted into each other (possibly increasing the number of states or transitions) and can therefore be seen as equivalent FSA representations. For the uniform modeling approach of this thesis, the Moore variant is selected, because this corresponds to the traditional form of representing HMM in speech recognition. Therefore, every state $s \in S$ of our FSA carries a symbol $\sigma(s) \in \Sigma$. The alphabet Σ also contains the empty symbol ϵ.

FSA are processed by walking from state to state along transitions, producing (or consuming) symbols on the way. The walk starts at the initial state and is complete when arriving at the final state. Complete walks are called *paths*. The formal language represented by a FSA is defined as the (possibly infinite) set of symbol sequences corresponding to all possible (different) paths through the FSA.

The main use of FSA is to check if they accept a given input symbol sequence, i.e. if there is a path through the FSA which produces the same symbol sequence. Therefore, the basic form of FSA is also called *finite-state acceptor*. If there is more than one possible path for any input symbol sequence, the FSA is denoted as *non-deterministic finite-state automaton* (NFSA), otherwise as *deterministic finite-state automaton* (DFSA). The notion of determinism generally applies to abstract machines.

FSA have a number of properties which render them attractive for practical use. An important property is that the problem of optimizing FSA, i.e. finding an equivalent automaton with the least number of states, is decidable. The optimization operation is also called *minimization*. For more expressive types of abstract machines, as pushdown automata, this problem is not decidable. FSA minimization is only possible for DFSA. Fortunately, NFSA can generally be converted to equivalent DFSA, in the worst case with an exponential rise in the number of states. Minimization of FSA is attractive because the resulting automaton has no redundancy. Hence, it can be processed efficiently, with minimum time and space requirements. Another property of FSA is that a number of unary and binary logic operations, similar to regular expression operations, can be carried out directly on the automaton level. Concatenation for example combines FSA in series, union combines them in parallel, intersection retains the common paths of both input FSA and Kleene closure enables arbitrary repetition of an automaton.

For applications involving insecure or statistical data extended versions of FSA exist, which carry weights on their transitions. These automata are called **weighted finite-state automata** (WFSA). Reusing the notations for FSA, a weighted finite-state automaton W can be defined as a tuple $W = (\Sigma, S, T, i, f)$ with a symbol $\sigma(s) \in \Sigma$ on each state $s \in S$ and a weight $w(t)$ on each transition $t \in T$.

Together with the introduction of transition weights, WFSA require a definition how to treat these weights, i.e. an appropriate calculus. Specifically, it needs to be decided how to combine the weights *along a path* and *between paths*. In speech recognition applications, it is desirable to view transition weights as log-likelihood values. For this notion, the *tropical semiring* $(+, min)$ is a suitable calculus, which implies that weights are summed up along a path, and weights on parallel paths are combined by applying a minimum operation. This means that if there are multiple paths through a WFSA for a particular input sequence, the best path is determined as the one yielding the minimum accumulated weight.

WFSA can be seen as a generalization of FSA. WFSA have similar properties as their unweighted counterparts. In particular, minimization of WFSA and automata operations are of special interest. It must be considered, however, that not every non-deterministic WFSA can be determinized directly. Yet it is often possible to modify such automata so that determinization becomes possible [Moh97].

In recent years, another class of finite-state machines has received increasing attention from the speech processing community, namely **weighted finite-state**

transducers (WFST) [MPR02]. WFST are a generalization of FSA and WFSA (and FST, the unweighted variant of WFST). In contrast to their acceptor counterparts, transducers do not only recognize input symbol sequences, but also map them into output symbol sequences. The transduction is carried out by assigning each state s (in the case of Moore machines) a tuple $(\sigma^i(s), \sigma^o(s))$, consisting of an input symbol $\sigma^i(s) \in \Sigma$ from the input alphabet Σ^i, and of an output symbol $\sigma^o(s) \in \Sigma^o$ from the output alphabet Σ^o. Apart from this, a WFST Θ is defined, similar to WFSA, as a tuple $\Theta = (\Sigma^i, \Sigma^o, S, T, i, f)$ with weights $w(t)$ on all transitions $t \in T$.

Many of the operations on FSA and WFSA also apply to WFST. Moreover, some operations only apply to transducers, such as *inversion*, which interchanges input and output languages. Of particular relevance to speech processing is an automata operation specially devised for WFST (and FST), called *composition*. The composition $X \circ Y$ of two WFST X and Y yields a new WFST which translates the input language of X into the output language of Y. A prerequisite for the composition operation to succeed is that the output alphabet of X and the input alphabet of Y are identical, and that the intersection of the respective languages is not empty.

The composition operation renders WFST particularly suitable for multi-level processing, where the whole string transduction problem can be split up into in a number of smaller transduction tasks. For speech recognition, imagine a WFST H mapping HMM states to phonemes, a second transducer D mapping phonemes to words, and a third transducer L containing the possible word transitions (and mapping them to themselves). From these components, a speech recognition transducer R, mapping HMM states to words, can be generated automatically through composition:

$$R = H \circ D \circ L$$

In certain cases (see [Moh97]) the minimization operation can be applied, so that R is optimized *globally* before decoding. Note that WFST minimization relies on the ability to shift input and output symbols within the transducer. Hence, it cannot be guaranteed that the 'atomic' transductions, i.e. the direct symbol mappings on the states or transitions, remain unchanged.

It is generally desirable to perform the composition of transducers off-line, so that a part of the computational work required for decoding is carried out off-line, resulting in faster on-line processing. However, off-line composition also means that all possible input strings, most of which don't occur during on-line processing, are preprocessed. This may yield final transducers whose size is too large for the available memory resources. Hence, transducer composition has a trade-off between CPU and memory load. A flexible handling of this trade-off has become possible lately with the introduction of on-demand automata operations such as composition or determinization [RPC95]. With this principle, the transducer composition is only carried out for those parts needed to process specific input strings, i.e. it is performed on-line. Hence, it can be transparently decided which compositions are carried out

off-line and which are left for on-line processing. It must be considered, however, that on-demand composition fundamentally increases the complexity of the otherwise rather straight-forward WFST decoder.

Under the assumption that Chomsky Type-3 grammars offer sufficient expressive power, WFST are generally suitable as a uniform modeling framework for speech interpretation. Their suitability for automatic optimization is a powerful property. Yet, the consequences of this property need to be considered.

A hierarchical meaning representation requires that the uniform knowledge model contains several modeling levels with semantic information. When using a single transducer, the hierarchical knowledge needs to be encoded in its flat automaton structure. Let's assume that there are two semantic modeling levels, namely *word classes* and *semantic concepts*. If we name the corresponding WFST K and C, respectively, and denote L as the transducer containing possible transitions between concepts, we can build a speech interpretation transducer I by:

$$I = H \circ D \circ K \circ C \circ L$$

Without special provisions, however, I will map directly from HMM states to semantic concepts, without direct access to the sub-surface information, i.e. words and word classes. Hence, information about the needed sub-surface symbols needs to be encoded *implicitly* into I. This can be done by introducing special marker symbols, which announce start and end of a symbol sub-sequence and act like brackets. This has the effect that the final transducer I cannot be optimized as high as before.

Furthermore, the symbol shifting during WFST minimization breaks synchronity between input and output symbols, so that it becomes difficult to uncover the temporal word alignment. Yet this information can be vital for further processing, such as computing confidence measures [LFRT04]. Another consequence of the transducer composition and optimization, specifically in off-line mode, is a limitation of the ability to dynamically modify the transducer at runtime. This ability might be required, e.g. in order to adjust the weighting of grammar parts depending on the state of a dialogue system, or in order to dynamically add new words to the grammar.

Because of these difficulties, and also because of a lack of appropriate tools for on-demand transducer operations, the finite-state *transducer* paradigm is not exploited for the one-stage speech interpretation problem of this thesis. Instead, an explicitly hierarchical modeling approach is preferred. Yet, this approach still makes use of finite-state *acceptors*, by hierarchically combining them to a knowledge representation similar to a stochastic context-free grammar. This type of grammar is introduced in the following section. In our approach, which is presented in Section 2.3, FSA minimization still plays an important role for *locally* optimizing parts of the model hierarchy.

2.2.3 Context-Free Grammars

Context-free grammars (CFG) consist of rules of the form:

$$A \rightarrow \gamma$$

where γ is an arbitrary sequence of terminals and non-terminals from the finite alphabets Σ and N, respectively. Formally we write $\gamma \in (\Sigma \cup \mathsf{N})^*$, where $\cup$ denotes union and $*$ is the Kleene star. The *LHS* of a CFG rule consists of a single non-terminal symbol A. CFG are Chomsky Type-2 grammars and thereby have greater expressiveness than regular grammars. In the special case that recursion only occurs via the leftmost or the rightmost symbol of all rules' *RHS*, CFG have equivalent regular grammar representations. These types of CFG are attributed as being left-linear or right-linear, respectively.

CFG are traditionally used in computational linguistics for modeling syntactic structure. Like all non-stochastic grammars, CFG have no ability to decide which is the better parse in the case that several parses exists, i.e. to *disambiguate*. Since speech is highly ambiguous, disambiguation plays an important role in speech processing. For tasks requiring disambiguation, CFG have been extended to *stochastic context-free grammars* (SCFG). In general, stochastic grammars assign probabilities to parses, so that disambiguation of multiple parses becomes possible by choosing the most probable one. SCFG augment each rule with a conditional probability p:

$$A \rightarrow \gamma\ [p]$$

To ensure proper normalization, the sum of p over all rules with the same *LHS* must be 1. The probability of a parse tree is then computed as the product of p over all rules used to expand each node in the parse tree.

SCFG have two properties desirable for a uniform model for speech interpretation, namely stochastic and explicitly hierarchical knowledge representation. As discussed in the previous section, the latter enables full control over the decoding process, including access to sub-surface units and temporal alignment, and allowing dynamic model modification. Instead of the formal grammar notation, we prefer to represent SCFG as abstract machines, more precisely as hierarchies of weighted finite-state automata. Our one-stage decoder then directly operates on the abstract machine, utilizing a technique called token passing. As discussed in the previous section, the machine representation enables automatic optimization. Although our explicitly hierarchical modeling approach prevents global model optimization, local optimization of automata can still be performed. The machine-centric view provides a common platform for different modeling techniques such as data-driven and rule-based approaches. While a formal grammar notation could also serve this purpose, we prefer to use the 'final' form of representation, which is directly fed into the decoder, as common platform.

In order to discriminate between the 'original' finite-state calculus and our explicitly hierarchical approach, we use the term transition network rather than finite-state

automaton. More precisely, the term transition network denotes a Moore machine in this work, so that formal language symbols are attached to network nodes instead of transitions. Our uniform, hierarchical knowledge model for speech interpretation is consequently called weighted transition network hierarchy.

2.3 Weighted Transition Network Hierarchy (WTNH)

In the previous section we described the selection of a uniform modeling framework for speech interpretation, called **weighted transition network hierarchy** (WTNH). The fundamental structure and properties of this knowledge representation are presented in the first two parts of the current section. The third part describes the basic structure of the results of applying WTNH. In the final part of this section, an overview over the integration of various knowledge sources into WTNH is given.

2.3.1 Basic Structure

WTNH are similar to stochastic context-free grammars or stochastic recursive transition networks [HC96]. Thereby, each transition network corresponds to a grammar rule, and the hierarchical network structure corresponds to the dependencies between rules. These dependencies exist between non-terminal symbols on the right-hand side of production rules and identical left-hand side symbols of the same or other rules.

The *RHS* definition of a SCFG rule corresponds to the internal structure of a transition network. As mentioned in the previous section, we prefer the Moore representation of abstract machines. Hence, transition networks in the hierarchy contain two different types of nodes, namely *terminal nodes* and *non-terminal nodes*. These can be seen as counterparts of terminal and non-terminal formal language symbols at the *RHS* of SCFG rules.

In order to enable unambiguous referencing to transition networks, each network is assigned a unique symbol. Network symbols are the counterparts of *LHS* symbols of SCFG rules. The hierarchical dependencies in our uniform knowledge representation are therefore references from non-terminal network nodes to networks. These links can be established automatically from a set of transition networks by finding matching symbols of nodes and networks.

The correspondence between SCFG and weighted transition network hierarchies cannot be seen if SCFG can have rules with identical *LHS*, because in this case we couldn't assign unique symbols to transition networks, and hence couldn't refer unambiguously to them. Fortunately, rules with identical *LHS* can easily be merged by applying the union of their *RHS*. In this form, SCFG can be converted directly to weighted transition network hierarchies. As has been discussed in the previous section, regular expressions have equivalent finite-state automata representations. Hence, the *RHS* of convertible SCFG rules may consist of arbitrary regular expressions of terminal and non-terminal symbols, which specifically includes union

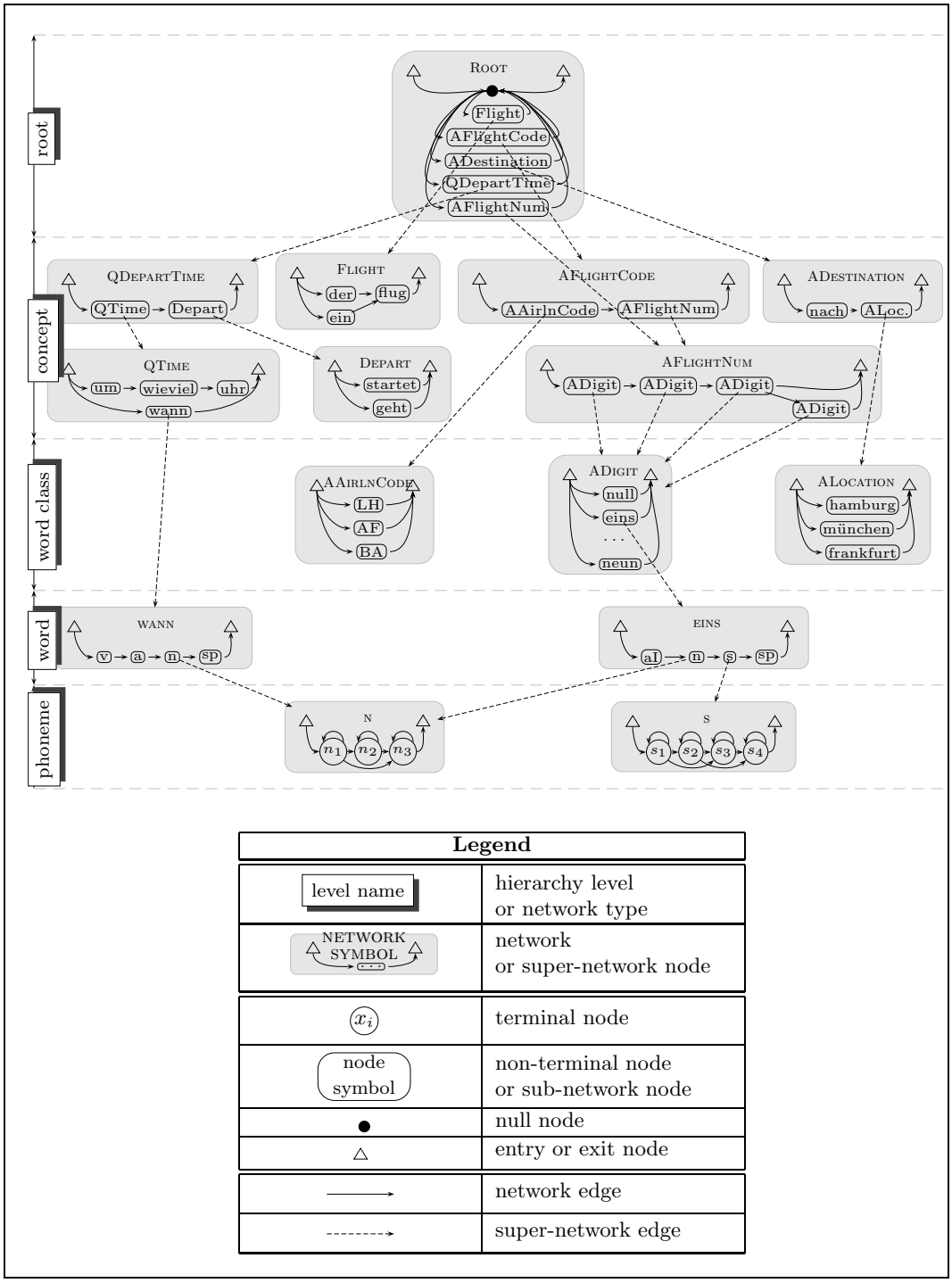
operations required for rule merging.

Figure 2.2 shows a simple example of a WTNH for the task of interpreting spoken language queries in an airport information system. For clarity of illustration, no transition weights are shown and the WTNH is only displayed partially, so that not all hierarchical dependencies end at a terminal node. A typical utterance contained in this WTNH is WANN STARTET DER FLUG L_H DREI SIEBEN NEUN EINS NACH HAMBURG (word-by-word translation: when starts the flight LH three seven nine one to hamburg). Figure 2.3 contains a CFG counterpart of the transition network hierarchy of Figure 2.2. The *RHS* rules of this CFG make use of the regular expression operators which were given earlier in Table 2.2 (see Section 2.2.2).

In Figure 2.2, each gray box contains a transition network. The bold-faced designator at the top of a gray box is the unique *network symbol*. Apart from this, a gray box contains the network representation in the shape of a directed graph. Terminal and non-terminal nodes are represented as circles or round-cornered boxes around their unique *node symbol*. For terminal nodes, which usually correspond to HMM states, the node symbol always has the form x_i, where x corresponds to the network symbol and i denotes the ith HMM state.

Because a non-terminal node refers to a subordinate network in the hierarchy, it is also denoted as *sub-network node*. In this context, the referenced network is called *sub-network*. In addition to terminal and non-terminal nodes, a transition network may contain *null nodes*. These are represented as bullets in Figure 2.2, and are e.g. contained in network ROOT. Null nodes correspond to the empty symbols ϵ of formal language. We assume that each network has exactly one *entry node* and one *exit node*, an arbitrary number of null nodes and at least one terminal or non-terminal node. Entry and exit nodes are special null nodes without incoming or outgoing arcs, respectively. They are represented as triangles in Figure 2.2. Transitions between nodes of a network are also called *network edges* and depicted as arrows in the example.

Viewing transition networks themselves as network nodes and references from non-terminal sub-network nodes to their sub-networks as edges yields the so-called *super-network*. In this context, we denote transition networks as *super-network nodes* and references from sub-network nodes to their sub-networks as *super-network edges*. The latter are depicted as dashed arrows in Figure 2.2. The super-network describes the hierarchical dependencies between the networks and is thus a representation of the hierarchical modeling structure. The super-network has a unique start node, which is also called *root network*, and one or several end nodes. The end nodes are those networks that contain no non-terminal nodes (networks N and S in Figure 2.2). A super-network is acyclic if the corresponding SCFG is non-recursive. Please note that acyclic super-networks may still possess cycles *within* transition networks (as is the case in networks ROOT and N in Figure 2.2).



Legend	
level name	hierarchy level or network type
NETWORK SYMBOL	network or super-network node
x_i	terminal node
node symbol	non-terminal node or sub-network node
●	null node
△	entry or exit node
⟶	network edge
⇢	super-network edge

Figure 2.2: *Example of WTNH, which contains the utterance* WANN STARTET DER FLUG LH DREI SIEBEN NEUN EINS NACH HAMBURG. *Transition weights are not shown.*

$$
\begin{aligned}
\textsc{Root} &\rightarrow (\text{Flight} \mid \text{AFlightCode} \mid \text{ADestination} \mid \text{QDepartTime} \mid \text{AFlightNum})^* \\
\textsc{QDepartTime} &\rightarrow \text{QTime Depart} \\
\textsc{Flight} &\rightarrow (\text{der} \mid \text{ein}) \text{ flug} \\
\textsc{AFlightCode} &\rightarrow \text{AAirlnCode AFlightNum} \\
\textsc{ADestination} &\rightarrow \text{nach ALocation} \\
\textsc{QTime} &\rightarrow \text{um wieviel uhr} \mid \text{wann} \\
\textsc{Depart} &\rightarrow \text{startet} \mid \text{geht} \\
\textsc{AFlightNum} &\rightarrow \text{ADigit ADigit ADigit ADigit?} \\
\textsc{AAirlnCode} &\rightarrow \text{LH} \mid \text{AF} \mid \text{BA} \\
\textsc{ADigit} &\rightarrow \text{null} \mid \text{eins} \mid \ldots \mid \text{neun} \\
\textsc{ALocation} &\rightarrow \text{hamburg} \mid \text{münchen} \mid \text{frankfurt} \\
\textsc{wann} &\rightarrow \text{v a n sp} \\
\textsc{eins} &\rightarrow \text{aI n s sp} \\
\textsc{n} &\rightarrow {n_1}^+ \; {n_2}^* \; {n_3}^+ \\
\textsc{s} &\rightarrow {s_1}^+ \; ({s_2}^+ \; {s_3}^* \; {s_4}^+ \mid {s_3}^+ \; {s_4}^+)
\end{aligned}
$$

Figure 2.3: *Context-free grammar representation corresponding to transition network hierarchy of Figure 2.2.*

2.3.2 Hierarchy Levels

The speech interpretation task comprises several levels of processing, such as acoustic-phonetic, lexical and semantic processing. Although all these processing levels are integrated into a uniform knowledge model and decoded simultaneously in a one-stage process in this work, it is still desirable to separate them logically in order to be able to assign different attributes to them. This is e.g. useful for applying different structural constraints or search parameters at different logical processing levels. For example, we restrict word classes to unions of single words, and language model factors to be only applied at the word level and above (see Section 5.9.1).

In order to reflect the logical processing levels in the WTNH, we attribute different *network types* to adjacent groups of transition networks. The assignment of network types is constrained by the requirement that the WTNH forms a proper type hierarchy. Consequently, the WTNH consists of different *hierarchy levels*, whose names correspond to the types of the networks on that level. By definition, the WTNH forms a proper type hierarchy if all super-network edges lead to a network on the next lower hierarchy level. This notion is illustrated in the example network hierarchy of Figure 2.2, which consists of 5 hierarchy levels. From the lowest to the highest level these are the phoneme level, the word level, the word class level, the concept level and the root level. The root level is a special hierarchy level which always needs to be present and must contain exactly one network, the so-called *root network*.

The introduction of hierarchy levels severely restricts the freedom of knowledge representations if the corresponding WTNH must form a proper type hierarchy. In order to enable more flexible and modular hierarchical modeling, we relax these

constraints in two ways:

- hierarchy levels may possess an arbitrary number of sub-levels
- certain hierarchy levels may be skipped

Consequently, the type hierarchy validity check needs to be modified. In order to allow hierarchy levels to be composed of an arbitrary number of sub-levels, a super-network edge may, in addition to the next lower hierarchy level, also lead to a network on the same level. In Figure 2.2, this is e.g. the case between networks AFLIGHTCODE and AFLIGHTNUM, which both reside on the semantic concept level.

Level skipping occurs in two variants, namely skipping of sub-levels and skipping of entire hierarchy levels. The latter is only allowed if a super-network edge may lead to a network on the next but one lower hierarchy level (as between networks QTIME and WANN, where the word class level is skipped). Skipping of sub-levels can be useful at the semantic level, for example, so that a concept may appear both at the surface and as part of a higher conceptual category. An example for this is the concept AFLIGHTNUM in Figure 2.2, which appears both in the root network and in the concept AFLIGHTCODE.

2.3.3 Semantic Trees

After introducing the basic structure and properties of our uniform knowledge model, the outcome of applying such a model to analyze spoken utterances is now considered. While the one-stage decoding process will be discussed later in greater detail, it is already clear from Section 2.2.2 that the decoder needs to find the best, i.e. least-cost, path through a WTNH for a given sequence of acoustic observations. From the best path, the final decoding result is extracted in the shape of an ordered, labeled tree.

A valid path is generated by walking from node to node along node transitions, beginning with the entry node of the unique root network and ending at the exit node of the root network. Due to the hierarchical model structure, a walk leading into a (non-terminal) sub-network node continues its way at the entry node of the referenced sub-network. In other words, the path walks along a super-network edge. On the contrary, walks arriving at an exit node of a non-root network continue at the sub-network node they previously descended from, so that they traverse a super-network edge backwards. When visiting a terminal node, a path consumes the next acoustic observation in the input sequence. Hence, a path arriving at the exit node of the root network is only valid if it has consumed all given observations.

While walking along a path, recording the symbols of the traversed networks yields an ordered tree with those symbols at its tree nodes. Such a tree represents the hierarchical dependencies between the visited networks and the order in which they were traversed. If we view the network hierarchy as a stochastic context-free grammar, a decoded tree corresponds to a parse tree of the grammar.

Usually, the HMM states and phonemes are not utilized for further processing. Therefore, only networks on the word level and above are recorded. As the result-

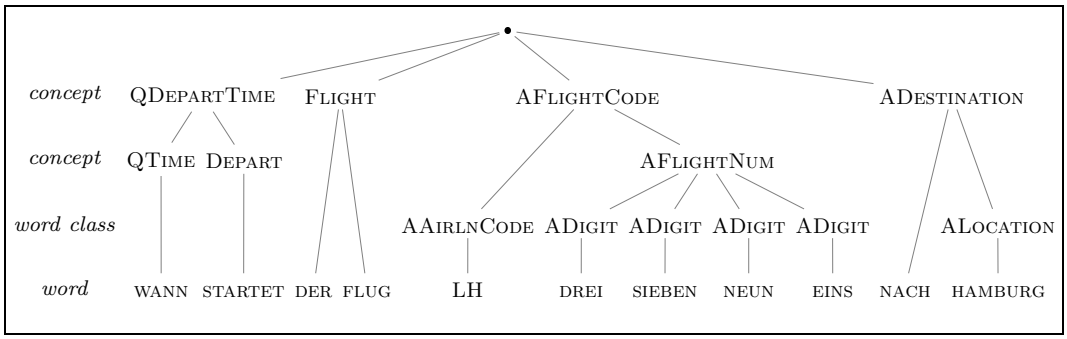


Figure 2.4: *Semantic tree for the example of Figure 2.2.*

ing trees carry symbols with semantic meaning, we denote them as **semantic trees**.

Figure 2.4 depicts the semantic tree of Figure 2.2 for the word sequence wann startet der flug LH drei sieben neun eins nach hamburg. We generally draw semantic trees so that a horizontal line through the tree touches only tree nodes of the same type. The tree node types correspond to the network types and to the hierarchy levels of the WTNH, which were described in the previous section. In Figure 2.4, they are shown at the left side next to the tree. Note that there may be less hierarchy levels in a semantic tree than in the WTNH the tree is derived from. When drawing nodes of a type with one or more sub-levels, the node is placed on the highest possible (sub-)level. For example, the tree node Flight is drawn on the upper concept level in Figure 2.4, not on the lower one.

A semantic tree always has a unique root node which corresponds to the unique root network of the WTNH. Because of its uniqueness, we neither show its tree node symbol nor its type throughout this thesis, but only a black dot. However, for the tree-based evaluation measure defined in Chapter 4, it is important that the semantic trees are rooted trees, i.e. that they possess exactly one root node.

The network hierarchies examined in this thesis always have one fully occupied word level, because word level skipping does not occur. Therefore, the leaf nodes of the resulting semantic trees are always words. However, this is no general necessity from the viewpoint of knowledge representation, decoding or evaluation. It might not even be required from a modeling point of view, if one does not assume that words must always be the fundamental units of meaning representations.

2.3.4 Knowledge Representation within WTNH

After having defined the basic structure of WTNH, an overview is now given of how the different knowledge sources for speech interpretation can be integrated into them. In particular, we focus on the representational limitations of WTNH in this section. At first, acoustic-phonetic and lexical modeling is discussed briefly. Although these models are not a focus of this thesis, their quality should reflect the state-of-the-art

of speech interpretation systems, so that the experimental results gained in this work can be judged adequately. In the last part of this section, meaning representations and their use for WTNH are considered.

Hidden-Markov Models

The acoustic-phonetic model (AM) is a fundamental component of a speech recognition or understanding system. Its quality and robustness is an important factor for the overall performance of the whole system. With the aid of this model, the mapping of sequences of acoustic observations to basic speech units is performed. Due to the large acoustic-phonetic variability and ambiguity of speech, the mapping is performed in a statistical manner, i.e. the likelihood of different mappings is estimated, so that the most likely one for the whole speech utterance can be chosen. One fundamental problem of acoustic-phonetic modeling is the variable length of speech units, which requires a dynamic model with respect to its temporal dimension.

Hidden-Markov Models (HMM) [RJ86, Rab89] have been the most widely used statistical AM for many years. A continuous HMM consists of a fixed set of states and transitions between those states. Each state has an associated likelihood function for consuming (or, depending on the viewpoint, emitting) acoustic observations. The likelihood functions are probability density functions (PDF). HMM states are traversed along their state transitions in temporal direction. A certain degree of temporal dynamics is achieved by allowing states to be traversed multiple times, or to be skipped. The former is realized by adding cycles to HMM states through self-transitions. The state transition function also has a statistical component, in the shape of transition weights. In addition to transitions between states, the definition of initial and final states and associated transition weights also belongs to HMM. The PDF parameters and the transition weights of HMM are estimated automatically from speech data.

The state-transition structure of a HMM can be reproduced straightforwardly with a weighted transition network, by replacing HMM states with network nodes and state transitions with node transitions. While the likelihood of HMM state transitions can be represented as transition weights in the WTNH, the emission likelihood computation of state PDF needs to be performed externally. In the example of Figure 2.2, typical examples for left-right HMM with 3 and 4 states are depicted at the phoneme hierarchy level. A left-right HMM consists of a sequence of states which are only traversed in one direction.

HMM have been used in speech recognition for modeling various speech units, including words, syllables, demi-syllables, triphones, diphones and monophones. The optimum unit choice is task-dependent and influenced by factors such as speaker dependence, vocabulary size, amount of training data, computational power and target language. For a German-language, speaker-independent, medium-size vocabulary application with fairly large amounts of training data and the computational power of current PC workstations, triphone HMM are a suitable choice. Triphones are context-dependent speech units, whose context spans the previous, the current

and the subsequent phoneme. If the triphone context is allowed to extend over word boundaries, they are called cross-word triphones. However, if the explicitly hierarchical structure of WTNH should be preserved during decoding, as is the case in this work, cross-word triphones cannot be used. The reason for this is that a path can enter different successors after leaving a word. Hence, the following phoneme could only be determined uniquely by looking ahead. This would require special treatment of the phoneme level during decoding, which contradicts the uniformity of the knowledge representation and processing scheme. Due to this limitation, intra-word triphones are used throughout this thesis, which do not perform as good as cross-word triphones [Six03]. Yet it should be noted that it is generally possible to cancel this limitation by converting WTNH to implicitly hierarchical representations, such as WFST. The WFST approach was already discussed in Section 2.2.2.

Pronunciation Lexicon

The transition from phonemes to words is carried out with a lexical pronunciation model. In the basic case, a pronunciation lexicon contains one phoneme sequence for each word, namely the canonical (standard) pronunciation of the word. Such a lexicon can be created manually, by automatic grapheme-to-phoneme conversion techniques, or by a combination of the two. In order to account for alternative pronunciations diverging from the standard, so-called pronunciation variants can be added manually or generated automatically. The latter usually requires morphological knowledge. Furthermore, different weighting of alternatives can be performed, e.g. through estimating weights from training data.

Adding many variants to the lexical model is especially useful if pronunciation varies fundamentally across the target users of the system, e.g. because of dialectal variation. However, it must also be considered that this effectively increases the confusability between words, and therefore may have a negative influence on the system performance if not used with care. Since a close examination of the lexical model should not be the focus of this thesis and as word pronunciation did subjectively not vary much across the recorded speakers, we refrained from systematically adding pronunciation variants to the lexicon of the test system. Instead, only some variants were added unsystematically for frequently occurring deviations from the canonical form. The canonical pronunciations were created manually. Due to the absence of automatic pronunciation generation and also because it was not used for our robust semantic analysis, the lexical model of this work does not contain morphological knowledge.

Hence, the pronunciation lexicon is represented within our uniform knowledge model by creating one transition network for each lexicon entry, and adding the phonemes of the canonical pronunciation as a linear node sequence (see Figure 2.2 for examples). Alternative pronunciations can be added as additional paths to the word network. Pronunciation weighting is generally possible by assigning corresponding transition weights.

In speech recognition, two different ways of representing lexical knowledge for

decoding are traditionally distinguished. A linear lexicon separately represents each word and its pronunciations, whereas a tree lexicon combines identical word beginnings in an unordered tree of phonemes, which can reduce the search effort during decoding [ON95]. A tree representation of lexical knowledge is not directly possible with WTNH, as each word requires its own transition network so that it can be referred to from higher levels of the hierarchy. Yet, automatic network minimization can be performed for each word separately, so that at least the different pronunciations of a word are represented in an optimized way. Moreover, as is discussed in Chapter 3, the one-stage decoder jointly propagates all search paths within a transition network, so that paths crossing the same phoneme within different words at the same time frame are treated together at the phoneme level.

Meaning Representation

As mentioned in the introduction of this work, the ultimate goal of speech understanding is the automatic transformation of spoken utterances to formal representations of the ideas transported with the utterances, in other words the utterances' meaning. This transformation can be viewed as the machine's process of 'understanding' or 'interpreting' user utterances. In this sense, speech serves as a medium to transport ideas between human and machine. The formal meaning representation provides the basic semantic structure that subsequent processing stages of the machine act upon, e.g. with the aim to give advice to users or to execute actions desired by users. In order to perform its task, the machine also needs a certain degree of world knowledge, whereby the 'world' of current speech understanding systems comprises only special fragments of the real world, e.g. an airport's flight database or a company's telephone directory.

The speech understanding problem is usually approached by subsequently converting speech to text and text to meaning. At first glance, these two processing steps can be carried out by combining automatic speech recognition with 'deep' semantic analysis techniques from the field of natural language understanding (NLU). This would for example lead to meaning representation such as Semantic Networks or First Order Predicate Calculus [RN95], where the semantic analysis is based on explicit morphological and syntactic analyses. However, such an approach would only be feasible if speech recognizers were able to produce nearly perfect transcriptions of speech utterances, on which the deep NLU processing was able to rely. Yet, this is rarely the case in practice today, especially if system users should be able to talk in a natural way and in natural situations. With the resulting erroneous and ungrammatical text, the deep syntactic analysis is likely to fail, as for example the German Verbmobil project [Wah00] has shown.

In order to cope with such kind of input, **robust semantic analysis** techniques need to be applied. The robustness is achieved by lowering the analysis depth on the one hand, and narrowing the complexity of application domains on the other hand. The former means that little explicit morphological and syntactic knowledge is used, or even omitted completely. The semantic analysis is directly performed on

the word sequence in this case. Such an approach is e.g. taken by so-called *semantic grammars* [BB75], which were originally developed for text-based dialogue systems. Combining syntax and semantics in a single 'semantic' model renders the model highly domain-specific. Hence, reusing models for other domains is hardly possible, so that new semantic models need to be developed for each new domain.

Another measure to increase robustness is a tighter integration of speech recognition and semantic analysis, aiming at avoiding certain types of speech understanding errors caused by early decisions. This kind of approach is chosen in this thesis, by creating a uniform knowledge representation for both speech recognition and semantic modeling, and utilizing this for a one-stage decoding process.

As a downside of the tightly integrated modeling approach, certain processing steps of semantic analysis are hard to carry out, e.g. canonical representation or consideration of background knowledge. Because of these limitations, we rather use the less ambitious term *speech interpretation* instead of speech understanding. In the remaining parts of this section, the properties of meaning representation within our one-stage speech interpretation framework are discussed.

In natural language processing, meaning is typically represented through discrete symbols, denoted as **semantic objects**, and relations between those objects. For robust semantic modeling, words can be viewed as the fundamental (terminal) semantic objects. Following the principle of compositionality, we assume that the meaning of an utterance is composed from the meanings of its parts. From an atomic point of view, this principle implies that a composed meaning of a group of semantic objects can be expressed through a new, superordinate object. Through its meaning, the new semantic object is related to its subordinate objects.

Generally, different semantic object groups can be attributed with identical meanings. Such a set of object groups which share a common meaning is called **semantic category**. Semantic categories are the basic building blocks of robust semantic models. The basic structure of our semantic model is a hierarchical combination of semantic categories. This implies that the meaning of an utterance can be represented as a graph of semantic objects.

When representing semantic objects in a hierarchical structure with SCFG or with WTNH, the grouping of objects is limited to consecutive items (or single ones), and preservation of order. Consequently, the meaning representation of an utterance is an ordered tree of semantic objects, which we call semantic tree (see Section 2.3.3).

In order to illustrate the notions of semantic objects and categories, we again refer to the example of Figure 2.2. In this WTNH, the semantic model is composed of three types of semantic objects, namely words, word classes and concepts. Each semantic category of the semantic model is implemented as a transition network. The members of a category correspond to the symbol sequences contained in the paths through the transition network. At the word class level, the category members are single words by convention. At higher levels, the members may also be sequences of semantic objects. The name of a transition network hints at the meaning shared by its category members. For example, the category ALocation

contains semantic objects denoting locations (of airports), AFLIGHTNUM contains valid flight numbers composed of three or four consecutive digits, and QTIME subsumes words and phrases indicating a temporal question. The root network is a special category whose members don't share a special meaning as such, except that they all represent valid sequences of top-level objects.

The categorization of semantic objects can express different kinds of semantic relations between the category object and its members. In this thesis, we discriminate between three basic types of semantic categories. The semantic objects contained in the first type of category are related to the pieces of information that the dialogue system can give advice about. In the domain examined in this thesis, these are the items in the airport database, e.g. airport locations, arrival and departure times, flight codes and airline names. The names of such semantic objects are prefixed with an 'A', such as in ALOCATION or AFLIGHTCODE. The second type of category is based on words or phrases suggesting a question, such as ZU WELCHER ZEIT (word-by-word translation: at what time) or WO (where). These semantic objects are prefixed with a 'Q', such as in QTIME or QLOCATION. All other semantic objects are subsumed in a third type of category, whose names have no special prefix.

A property of representing meaning with WTNH is its significance of temporal order. While order is important on the acoustic-phonetic, lexical and syntactic modeling levels, ordering of meaning units only matters in certain cases, if at all. To illustrate this notion, consider the following textual utterances:

- FLUG LH DREI ACHT SIEBEN NACH HAMBURG
  (word-by-word translation: flight LH three eight seven to hamburg)
- FLUG NACH HAMBURG LH DREI ACHT SIEBEN
  (word-by-word translation: flight to hamburg LH three eight seven)
- FLUG LH DREI SIEBEN ACHT NACH HAMBURG
  (word-by-word translation: flight LH three seven eight to hamburg)

Clearly, the first two utterances have the same meaning, although the order of the phrases LH DREI ACHT SIEBEN and NACH HAMBURG is swapped. The third utterance corresponds to the first one, except that the digits ACHT and SIEBEN occur in different order. In contrast to the phrase change from the first to the second utterance, this digit change alters the utterance meaning, because the order of the digits is significant for the meaning of the flight number[1]. Considering Figure 2.5 reveals that the semantic trees corresponding to the three utterances are all different, although the first two utterances have identical meaning.

This example shows that the order significance of WTNH prevents that meaning is represented in a *canonical form*, so that the speech interpretation system does

[1]Please note that it would be possible to render the flight number category AFLIGHTNUM insignificant of order by explicitly discriminating its members, e.g. by replacing the three ADIGIT with ADIGIT1, ADIGIT2 and ADIGIT3.

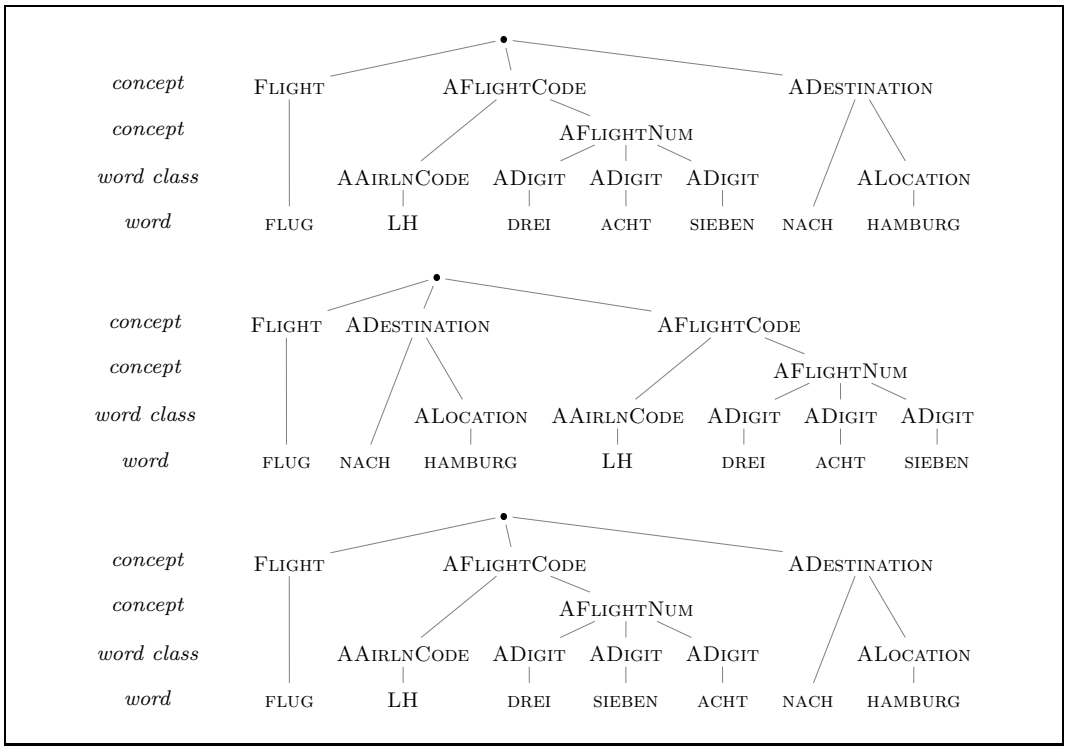


Figure 2.5: *Example of semantic trees with three different word orders and two different meanings.*

not necessarily produce identical output for different input utterances with identical meaning. Hence, the canonical form must be established in a postprocessing stage of the speech interpreter.

Another property of semantic trees is their inability to represent nested structures. This can be of importance if a semantic object consists of spatially separated objects. For example, consider the utterance WO KOMMT DER FLUG BA NEUN NULL SIEBEN SECHS AN (word-by-word translation: where comes the flight BA nine zero seven six at), where the members of the concept ARRIVAL, namely KOMMT and AN, are 6 words apart. In such a case it is impossible to assign the separated objects to one semantic category without also assigning all embraced words to the same category. If this is not desired, the semantic category can only be split into two parts, which are rejoined at a later processing stage. However, a link between the separated objects can still be established if the dependencies within the semantic model are capable to bridge the gap between them (see also Section 5.8).

Chapter 3

A One-Stage Decoder for WTNH

This chapter presents a description of the basic one-stage decoding scheme for the uniform modeling framework discussed in Chapter 2. The decoder, whose basic principle was also described briefly in [TFLR03a], is called *One-stage Decoder for Interpretation of Natural Speech* (ODINS). Please note that the experimental results presented in this thesis were performed with an optimized decoder implementation that differs in some aspects from the basic scheme described here. The optimizations, which are not a part of this thesis, mainly improve the runtime behavior of ODINS. Yet, nor do these modifications alter the basic decoding scheme, neither the presented experimental results.

This chapter is organized as follows: In Section 3.1, the hierarchical search problem is formulated in a generalized way. Section 3.2 discusses how the hierarchical search problem can be solved with a time-synchronous decoding scheme based on the token passing principle, and compares the proposed procedure with other similar decoding approaches. Finally, Section 3.3 presents the results of an experiment which quantifies the theoretical advantage of the one-stage decoding strategy over a corresponding two-stage approach.

3.1 Hierarchical Search Problem

The underlying assumption of the hierarchical modeling approach of Chapter 2 is, that a sequential correspondence exists between the input symbols $\mathbf{S}^i$ and the output symbols $\mathbf{S}^o$ of the decoder [PLV93]. More specifically, if $\mathbf{S}^i$ consists of a sequence of $|\mathbf{S}^i|$ symbols $s^i_1, s^i_2, \ldots, s^i_{|\mathbf{S}^i|}$ and $\mathbf{S}^o$ consists of a sequence of $|\mathbf{S}^o| \leq |\mathbf{S}^i|$ symbols $s^o_1, s^o_2, \ldots, s^o_{|\mathbf{S}^o|}$ a sequential correspondence means that $\mathbf{S}^i$ can be segmented into $|\mathbf{S}^o|$ consecutive sub-sequences $\mathbf{S}^i_1, \mathbf{S}^i_2, \ldots, \mathbf{S}^i_{|\mathbf{S}^o|}$, so that each input sub-sequence $\mathbf{S}^i_k$ directly corresponds to an output symbol s^o_k, where $k = 1 \ldots |\mathbf{S}^o|$.

Let's assume that in a speech recognition or speech interpretation system, $\mathbf{S}^i$ are

the acoustic observations and $\mathbf{S}^o$ is a sequence of words or a sequence of semantic objects. Then, the task of the decoder can be expressed as the problem of finding the optimum output sequence for a given input sequence. As the decoder decisions should be based on a statistical model, 'optimum output sequence' is rephrased as 'most probable output sequence'. Hence, using the definition from above, the search problem can be formulated as finding the output sequence $\mathbf{S}^{o*}$ for which the statistical model, given an input sequence $\mathbf{S}^i$, yields maximum likelihood:

$$\mathbf{S}^{o*} = \arg\max_{\mathbf{S}^o} P(\mathbf{S}^o|\mathbf{S}^i) \tag{3.1}$$

Due to the known difficulty of directly estimating the a-posteriori probability $P(\mathbf{S}^o|\mathbf{S}^i)$, Equation 3.1 is rewritten with Bayes' formula as:

$$\mathbf{S}^{o*} = \arg\max_{\mathbf{S}^o} \frac{P(\mathbf{S}^i|\mathbf{S}^o)P(\mathbf{S}^o)}{P(\mathbf{S}^i)}$$

As the denominator $P(\mathbf{S}^i)$ doesn't depend on an argument of the maximum operation, it can be omitted, yielding:

$$\mathbf{S}^{o*} = \arg\max_{\mathbf{S}^o} \Big[P(\mathbf{S}^i|\mathbf{S}^o)P(\mathbf{S}^o)\Big] \tag{3.2}$$

While this equation describes the search problem for the case where the output only consists of a flat symbol sequence, our speech interpretation task requires hierarchical output and modeling. Therefore, we extend the notion of sequential correspondence from above in a general way by allowing an arbitrary number of intermediate levels of representation. This can be achieved by assuming pairwise sequential correspondence between adjacent levels in the hierarchy. As a consequence of this assumption, the resulting symbol hierarchy forms a proper, ordered tree structure.

In order to describe this extended notion of the search problem formally, it is assumed for the moment that the tree structure has a constant height, corresponding to the total number of levels of representation. For this purpose, we denote $\mathbf{T}$ as an ordered, constant-height tree consisting of $L \geq 2$ hierarchy levels. By definition, a horizontal line through $\mathbf{T}$ touches all tree nodes belonging to a hierarchy level l of $\mathbf{T}$, with $1 \leq l \leq L$. The sequence of tree nodes contained in l is denoted as $\mathbf{T}^l$. Thereby, $l = 1$ is defined as the lowest hierarchy level which contains the leaf nodes of $\mathbf{T}$ or, in other words, the input symbols of the decoder. The level $l = L$ refers to the highest level, containing the decoder's output symbols. The levels $l = 2 \ldots (L-1)$ contain the symbols of the intermediate levels of representation.

More specifically, the ith tree node of $\mathbf{T}^l$ carries the symbol s_i^l, so that we can write $\mathbf{T}^l$ as a sequence of $|\mathbf{T}^l|$ tree node symbols:

$$\mathbf{T}^l = s_1^l, s_2^l, \ldots, s_{|\mathbf{T}^l|}^l \tag{3.3}$$

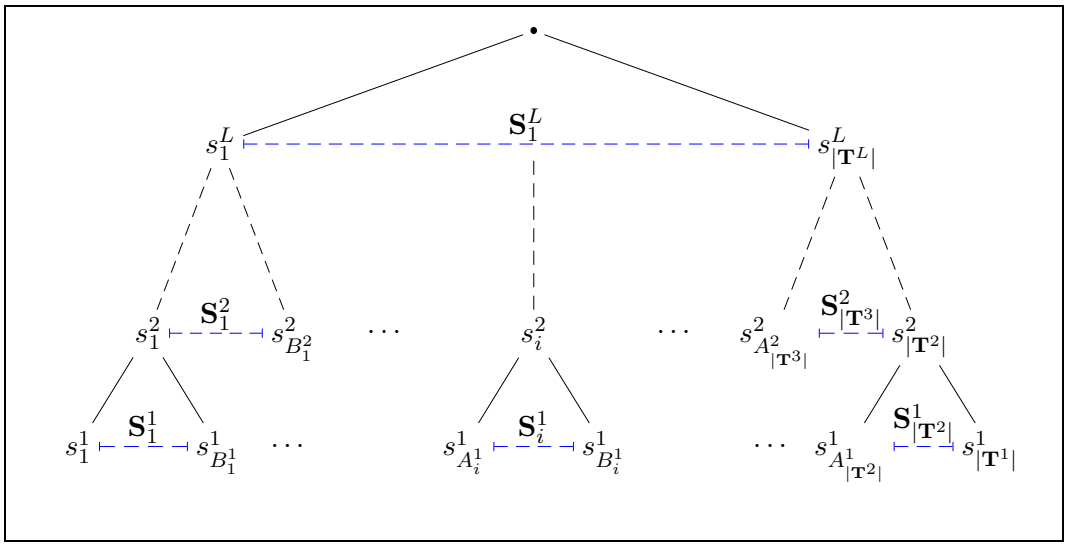


Figure 3.1: *Notation scheme for symbol tree structure.*

The tree structure of $\mathbf{T}$ implies that each level has at least as many nodes as the next higher level, so that:

$$|\mathbf{T}^1| \geq |\mathbf{T}^2| \geq \ldots \geq |\mathbf{T}^L|$$

Applying the notion of sequential correspondence to adjacent hierarchy levels yields that the tree nodes of level l can be split into $|\mathbf{T}^{l+1}|$ consecutive sub-sequences $\mathbf{S}_i^l$

$$\mathbf{T}^l = s_1^l, s_2^l, \ldots, s_{|\mathbf{T}^l|}^l = \mathbf{S}_1^l, \mathbf{S}_2^l, \ldots, \mathbf{S}_{|\mathbf{T}^{l+1}|}^l \tag{3.4}$$

so that each sub-sequence $\mathbf{S}_i^l$ directly corresponds to a tree node s_i^{l+1} on the next higher level. Figure 3.1 illustrates the notation scheme of the discussed tree structure. The indices of the first and last symbols of $\mathbf{S}_i^l$ are denoted A_i^l and B_i^l, respectively:

$$\begin{aligned} \mathbf{S}_i^l &= s_{A_i^l}^l, s_{A_i^l+1}^l, \ldots, s_{B_i^l}^l \\ \text{with} \quad A_1^l = 1, \quad A_i^l = B_{i-1}^l \quad &\text{and} \quad B_{|\mathbf{T}^{l+1}|}^l = |\mathbf{T}^l| \end{aligned} \tag{3.5}$$

In tree notation, s_i^{l+1} is called the *parent* of $\mathbf{S}_i^l$. Likewise, the components of $\mathbf{S}_i^l$ are called the *children* of s_i^{l+1}.

With this notation of the tree structure of symbol sequences, the search problem for a hierarchical model with L hierarchy levels can now be formulated. For this purpose, we replace the input symbols of Equation 3.2 with the lowest tree level $\mathbf{T}^1$, and the output symbols with the other tree levels $\mathbf{T}^2, \ldots, \mathbf{T}^L$. For the task of speech interpretation the decoder is not desired to output all tree levels $\mathbf{T}^2, \ldots, \mathbf{T}^L$, but

Hierarchy Level	Term	Tree Node Type	Tree Level
root	$P(\mathbf{T}^6)$	concept	$\mathbf{T}^6$
concept	$P(\mathbf{T}^5\|\mathbf{T}^6)$	sub-concept	$\mathbf{T}^5$
sub-concept	$P(\mathbf{T}^4\|\mathbf{T}^5)$	word class	$\mathbf{T}^4$
word class	$P(\mathbf{T}^3\|\mathbf{T}^4)$	word	$\mathbf{T}^3$
word	$P(\mathbf{T}^2\|\mathbf{T}^3)$	phoneme	$\mathbf{T}^2$
phoneme	$P(\mathbf{T}^1\|\mathbf{T}^2)$	HMM state	$\mathbf{T}^1$

Table 3.1: *Correspondences between the hierarchy levels of Figure 2.2 and the terms of Equation 3.6, and the symbol types contained in the tree level sequences.*

only semantically relevant symbols (see also Section 2.3.3). Therefore, the search problem is defined as finding the most likely output tree $\mathbf{T}^*$ consisting of the top M levels of the total symbol hierarchy, with $1 \leq M \leq L$.:

$$\mathbf{T}^* = \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \Big[P(\mathbf{T}^1|\mathbf{T}^2,\ldots,\mathbf{T}^L) P(\mathbf{T}^2,\ldots,\mathbf{T}^L) \Big]$$

Using the chain rule, this equation can be rewritten as:

$$\mathbf{T}^* = \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \Big[P(\mathbf{T}^1|\mathbf{T}^2,\ldots,\mathbf{T}^L) P(\mathbf{T}^2|\mathbf{T}^3,\ldots,\mathbf{T}^L) \ldots P(\mathbf{T}^{L-1}|\mathbf{T}^L) P(\mathbf{T}^L) \Big]$$

Under the assumption that the tree nodes $\mathbf{T}^l$ on level l only depend on the tree nodes $\mathbf{T}^{l+1}$ on the next higher hierarchy level, this becomes:

$$\begin{aligned} \mathbf{T}^* &= \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \Big[P(\mathbf{T}^1|\mathbf{T}^2) P(\mathbf{T}^2|\mathbf{T}^3) \ldots P(\mathbf{T}^{L-1}|\mathbf{T}^L) P(\mathbf{T}^L) \Big] \\ &= \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \left[P(\mathbf{T}^L) \prod_{l=1}^{L-1} P(\mathbf{T}^l|\mathbf{T}^{l+1}) \right] \end{aligned} \tag{3.6}$$

For the example speech interpretation system of Figure 2.2, we have $L = 6$ and $M = 4$. Table 3.1 shows which hierarchy levels of Figure 2.2 are described by which terms of Equation 3.6, and which types of symbols are contained in the levels of the hierarchy tree.

As already discussed in Section 2.3.2, it is desirable to extend the flexibility of the modeling hierarchy by allowing hierarchy levels to be skipped. In order to take this into account in the search problem formulation, we still assume that the hierarchy is fully occupied, i.e. that no level skipping occurs, by imagining a 'dummy' network consisting of a single sub-network node for each skip transition. In Equation 3.6, these dummy networks can be ignored because they have a production probability of one. This notion of hierarchy level skipping also explains why our representation of output trees, as shown in Figure 2.4, doesn't yield constant-height trees as described

in the beginning of this section.

Using Equations 3.3 and 3.4, we can further decompose Equation 3.6 into tree node symbols and their corresponding symbol sub-sequences:

$$\begin{aligned}\mathbf{T}^* &= \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \left[P(\mathbf{T}^L) \prod_{l=1}^{L-1} P(\mathbf{T}^l|\mathbf{T}^{l+1}) \right] \\ &= \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \left[P(\mathbf{S}_1^L) \prod_{l=1}^{L-1} P(\mathbf{S}_1^l, \mathbf{S}_2^l, \ldots, \mathbf{S}_{|\mathbf{T}^{l+1}|}^l | s_1^{l+1}, s_2^{l+1}, \ldots, s_{|\mathbf{T}^{l+1}|}^{l+1}) \right]\end{aligned}$$

By assuming that a sub-sequence $\mathbf{S}_i^l$ only depends on its parent symbol s_i^{l+1}, this becomes:

$$\begin{aligned}\mathbf{T}^* &= \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \left[P(\mathbf{S}_1^L) \prod_{l=1}^{L-1} P(\mathbf{S}_1^l|s_1^{l+1}) P(\mathbf{S}_2^l|s_2^{l+1}) \ldots P(\mathbf{S}_{|\mathbf{T}^{l+1}|}^l | s_{|\mathbf{T}^{l+1}|}^{l+1}) \right] \\ &= \underset{\mathbf{T}^{L-M+1},\ldots,\mathbf{T}^L}{\arg\max} \left[P(\mathbf{S}_1^L) \prod_{l=1}^{L-1} \prod_{i=1}^{|\mathbf{T}^{l+1}|} P(\mathbf{S}_i^l|s_i^{l+1}) \right] \qquad (3.7)\end{aligned}$$

A conditional likelihood term $P(\mathbf{S}_i^l|s_i^{l+1})$ of Equation 3.7 describes the probability that the symbols of a sequence $\mathbf{S}_i^l$ occur as children of a symbol s_i^{l+1}. The unconditional likelihood for the occurrence of the surface symbol sequence is represented by the term $P(\mathbf{S}_1^L)$.

In this work, the underlying likelihood distributions of $P(\mathbf{S}_i^l|s_i^{l+1})$ and $P(\mathbf{S}_1^L)$ are modeled by the set of transition networks contained in a WTNH. Thereby, the name of a non-root transition network is equivalent to the symbol s_i^{l+1}, and the symbol sub-sequence $\mathbf{S}_i^l$ corresponds to a path through the network. A path through the root network produces $\mathbf{S}_1^L$. The likelihood of a network path is computed as the product of the likelihood values of all involved transitions. At the phoneme level, the HMM state emission probabilities are also included in the likelihood computation. Hence, for a given acoustic observation sequence, the total likelihood of a path through the WTNH is computed according to Equation 3.7 as the product of all transition and emission likelihood values contained in the path. Using these notions, the hierarchical search problem becomes the task of finding the path through the WTNH yielding maximum total likelihood.

3.2 Token Passing

After formulating the hierarchical search task, this section discusses how this task can be performed by a time-synchronous, one-stage, iterative search procedure, which provides the core of ODINS. The basis for this decoding approach is the token passing paradigm [YRT89], which is well-known in the field of speech recognition. For example, the decoder of the Hidden-Markov-Model Toolkit (HTK) [YEK+02],

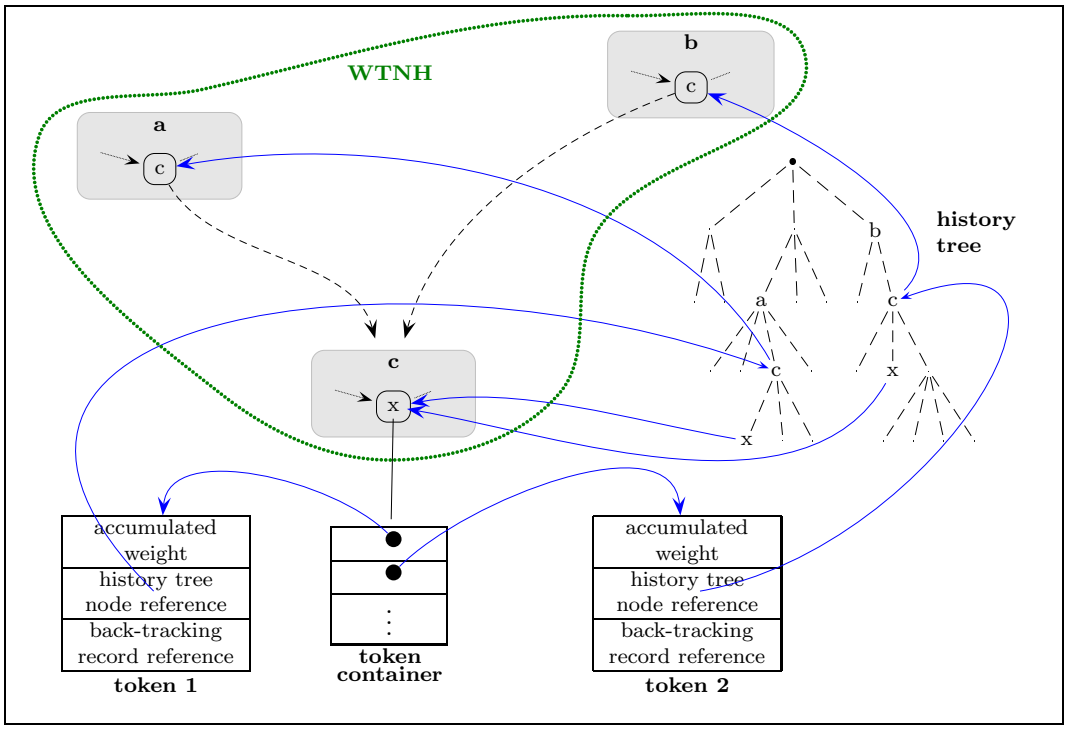


Figure 3.2: *Relations between WTNH, token containers, tokens and history tree.*

which is today a standard tool in speech recognition research, is based on a form of token passing.

With token passing, a time-synchronous, hierarchical Viterbi search can be performed by dynamic programming (DP) [SK83, Rus94]. The DP algorithm determines the best match between two patterns by finding the best path through a matrix of accumulated distances. For Viterbi decoding of HMM, the matrix is replaced by unfolding HMM states in temporal direction and explicitly specifying the possible state transitions in a so-called Trellis diagram. The decoding is performed in a time-synchronous way by computing the Trellis states column by column, from left to right. During this process, the principle of recombination is applied to minimize search effort, by recombining converging paths locally and only continuing with the best path at each Trellis state.

In [YRT89], Young formulates a generalized time-synchronous Viterbi decoding scheme, by representing each search path as a moveable token that is passed along the nodes of transition networks as time proceeds. The 'goodness' of a token is represented by the accumulated weight it holds. This weight is updated during token propagation with weights of traversed transitions and emission probabilities of visited HMM states. The time-synchronity of a token passing search is ensured by propagating each token over one HMM state only during each time frame. The

recombination principle is applied to token passing by merging tokens that meet at transition network nodes. The merging is performed by keeping only the token that holds the maximum score (log-likelihood), and discarding all others.

In the token passing algorithm described in [YRT89], Young assumes that none of the sub-networks are shared, so that there is a unique instance of each transition network for each reference to it. As also noted by Young, this implies that the super-network doesn't contain (direct or indirect) recursion, because this would require an infinite number of network instances. In order to eliminate this restriction in the algorithmic formulation presented here, the creation of unique network instances is avoided by directly operating on the WTNH (see Figure 2.2 for an example of a WTNH). This approach to token passing implies that network nodes are able to accommodate multiple tokens at once. We achieve this by associating a *token container* with each network node, which holds the set of tokens that currently reside at a specific network node. Figure 3.2 depicts the fundamental relationship between network nodes of WTNH, token containers, tokens and the token history tree (see next section).

As a consequence, it becomes possible to jointly propagate those token sets across transition networks. On the one hand, this reduces the computational effort in comparison to Young's approach. On the other hand, it becomes necessary to treat tokens separately with respect to recombination, because only those tokens sharing a common 'history' regarding the network hierarchy may be recombined. In the following section, we discuss how this requirement can be implemented by use of a so-called history tree.

3.2.1 History Trees

Instead of creating unique instances for referenced transition networks, we dynamically keep record of the 'ancestry' of tokens according to their way through the super-network. Since transition networks may contain multiple references to subordinate networks (see e.g. network AFlightNum of Figure 2.2, which repeatedly refers to network ADigit), references to sub-network *nodes* must be recorded. For a set of paths through a WTNH, those references to traversed sub-network nodes form an unordered tree. Since the ancestors of each tree node represent the history of a token regarding its current position in the network hierarchy, such a tree is denoted as *history tree*. The sequence of ancestor nodes of the history tree node where a token currently resides is called the token's *super-network history*.

Figure 3.3 shows the full history tree for all possible paths through the WTNH of Figure 2.2, except for the phoneme level. Since network AFlightNum is referred to directly from the root network and via AFlightCode, it occurs twice in the history tree. Hence, each digit, which is contained 4 times in AFlightNum, totally occurs 8 times in the history tree. Please note that if the super-network of a WTNH contains recursion, the corresponding full history tree becomes infinitely deep. Hence, if recursion is allowed, the history tree must be constructed dynamically during decoding.

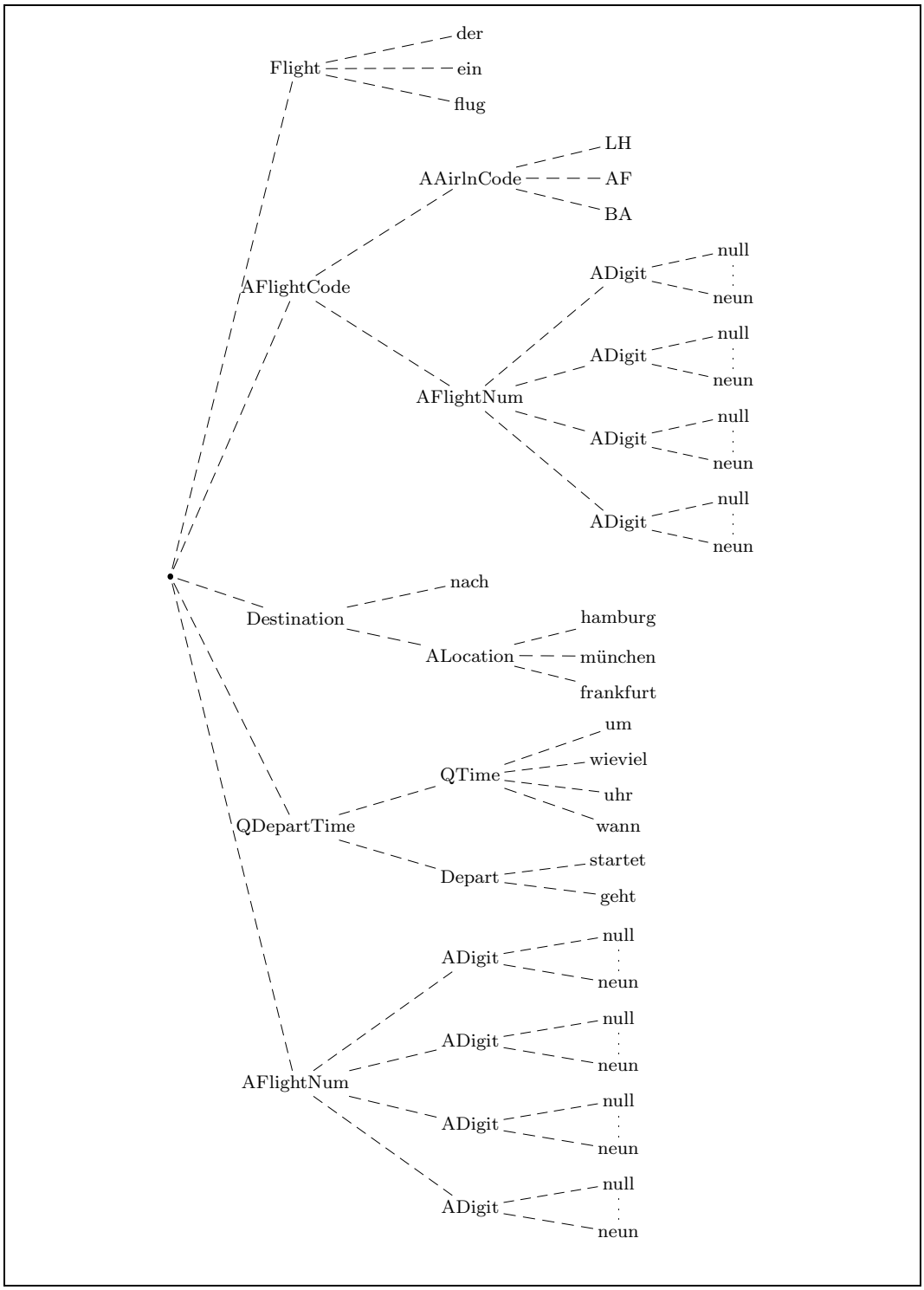


Figure 3.3: *History tree for the WTNH of Figure 2.2, without phonemes.*

As mentioned at the end of the previous section, history trees are a means to determine if tokens may be recombined. For this purpose ODINS uses a common history tree for the whole search process, and each token is augmented with a reference to a node of the common history tree (see Figure 3.2). This reference corresponds to the token's current position in the transition network hierarchy. Consequently, only tokens referring to identical history tree nodes may be recombined.

Figure 3.2 shows an example of two tokens that currently reside at node x of network c, and therefore are both placed in the same token container. Yet, these tokens may not be recombined due to their different super-network histories: Token 1 entered network c by descending from network a, whereas token 2 entered network c via network b. This fact is reflected by the token's history tree node references, which point to different history tree nodes that both carry identical symbols 'c'.

During token passing, the reference of a token to its history tree node must be kept up-to-date. This is accomplished by moving the reference to a child node when descending the WTNH to a sub-network, and by moving the reference to the parent tree node when ascending the hierarchy from an exit node. During ascension the history tree serves another purpose, namely to find the correct target sub-network node for token propagation. This target is determined from the parent of the token's current history tree node.

3.2.2 Token Propagation Scheme

As already explained in Section 2.3.3, a valid path through a WTNH is generated by walking from node to node along transitions, starting with the entry node of the root network and ending at its exit node. Hence, a token passing search is initialized by injecting a single token with score zero into the root network's start node. Generally, a token is propagated along a transition leaving the current node by moving it to the successor node and adding the transition weight to its accumulated score. If a node has multiple outgoing transitions, copies of the token are passed to all successor nodes, accordingly. When multiple tokens with the same super-network history meet at a network node, they are recombined as discussed in Section 3.2.1. The following special cases need to be considered during token propagation:

- When a token enters a sub-network node, it is moved to the start node of the corresponding sub-network. Hence, the token changes the super-network node. During this process, the token's history tree node reference is updated as explained in Section 3.2.1.
- When a token reaches the end node of a network, it is propagated back to the sub-network node it once descended using knowledge from the history tree as detailed in Section 3.2.1. More precisely, the token is propagated along the transitions leaving this sub-network node. Hence, the token traverses the super-network in backwards direction during this step.
- When tokens are passed to a terminal node, the HMM state's emission score

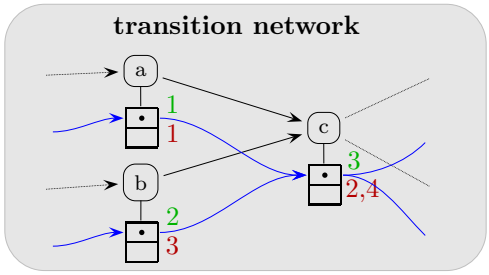


Figure 3.4: *Token recombination for in-order processing (numbers (1,2,3) above arcs) and out-of-order processing (numbers (1,2,3,4) below arcs).*

for the current acoustic observation is added to the token score as well as the transition score.

As explained in the beginning of Section 3.2, our token passing scheme directly operates on WTNH, so that network nodes generally accommodate a set of several tokens at once, which can be propagated simultaneously. In return, it is desirable for efficient processing to visit network nodes as rarely as necessary. This can be achieved by processing network nodes in topological order if possible, i.e. if the network is acyclic. Another reason for topological-order processing is, that it enables to prevent late recombination of tokens by letting converging tokens meet as early as possible, and hence avoid redundant computations. This cannot be ensured if stepwise token propagation is performed out-of-order.

Figure 3.4 demonstrates the advantages of in-order network processing: In the topologically sorted transition network, node c is processed after nodes a and b, so that tokens are recombined at the earliest possible point, i.e. at node c. In contrast, if node c is processed before node b, the token passed via node a has already left node c when the token passed via node b arrives there. Hence, the two tokens don't meet at node c, so that their recombination is delayed. Moreover, node c needs to be processed a second time in this case.

In contrast to our approach, the ISIP ASR system [JZP+01], which utilizes a similar one-stage decoding method with token passing as ODINS, uses so-called 'traces' to avoid late recombination. A trace of a token is a recording of the node sequence the token has passed in previous propagation steps. When a token comes across a trace of another token, token recombination can be carried out immediately, instead of waiting until the tokens themselves actually meet. Yet, this solution is not optimal because it doesn't avoid the propagation steps that the earlier token has carried out after leaving the optimum point for recombination (node c in Figure 3.4).

Similar to within-network processing, it is also desirable to process the whole network hierarchy in an optimum way. For this purpose, it needs to be considered

how time-synchronity is ensured during the token propagation process. As already briefly mentioned in Section 2.3.3, time is only consumed during decoding when terminal nodes are visited and the emission score of the next acoustic observation of the preprocessed input speech is added to the token score. Hence, the decoding is time-synchronous if each (surviving) token visits exactly one terminal node during each time frame. Similar to the ISIP system, we therefore perform token propagation iteratively by dividing each time frame into three phases:

Phase 1: Starting at the root network, tokens are propagated downward until all of them reside at a terminal network node.

Phase 2: Each token is propagated exactly once across the current terminal node, consuming a time frame and a corresponding emission score.

Phase 3: Tokens are propagated upwards through the network hierarchy, from the lowest hierarchy level to the root network.

During the 'timeless' phases 1 and 3, it is sufficient to process each network only once if processing occurs in a correct order. Similar to within-network processing, topological sorting of the super-network leads to a correct ordering. Hence, transition networks are processed in topological order during Phase 1, and in reverse topological order during Phase 3, whereas during Phase 2 networks containing terminal nodes can be processed in arbitrary order. If a WTNH contains recursion the super-network is cyclic, so that its topological sorting becomes impossible. In this case, networks need to be processed more than once during Phase 1 and Phase 3.

In order to further reduce the computational effort for decoding, different pruning techniques can be applied to concentrate search on the most promising decoding hypothesis by discarding tokens with low scores early. In speech recognition, a standard pruning technique is beam pruning, where all search paths whose scores are worse than a constant threshold from the token with the best score are discarded. This process is normally repeated at each time frame. Thereby, search is only carried out on a 'beam' of most promising hypotheses. The number of hypotheses surviving a beam pruning step may vary significantly within and across utterances, depending on the degree of ambiguity currently present. Generally, at instances of high ambiguity more hypotheses are considered than when ambiguity is low. Therefore, beam pruning avoids search errors, which occur if the globally best search path is erroneously pruned, while reducing the search effort by a considerable amount.

However, the dynamics of beam pruning can also be undesirable, because this may result in high variations in processing time, which for instance irritates system users. Therefore, more advanced pruning techniques, e.g. maximum instance pruning as in the ISIP system or confidence measure controlled pruning as in [FLRT05] can be applied. As runtime performance optimization is not a focus of this thesis, only beam pruning is considered here. Beam pruning is applied in Phase 2 of the token passing algorithm, when all tokens have settled on a terminal node. After

determining the best scoring token, the current beam width is computed by subtracting the predefined threshold from this score. Finally, all tokens with inferior score are discarded.

After the last acoustic observation of the input speech utterance has been consumed and the remaining tokens have been propagated upwards to the exit node of the root network, the decoding result needs to be extracted from the best token by back-tracking. This step is carried out, similar to [YRT89], via a linked list of so-called back-tracking records, which are created during forward propagation. Hence, each token holds a reference to a back-tracking record, as shown in Figure 3.2. Following the back-tracking records of the best token reveals the sequence of transition networks that the token has visited. This sequence is then converted into a semantic tree as described in Section 2.3.3.

3.2.3 Constrained Token Passing

After presenting a token passing based solution to the hierarchical search problem of ODINS, a modified version is discussed in the following. The modification enables decoding based on textual representation of utterances instead of speech input. The representation of textual input may be weighted and contain alternative hypotheses in the shape of a word lattice, which can be produced by speech recognition systems [DGP99]. Even more, the input may already be hierarchically structured like the so-called lattice hierarchy described in [LFRT04], which can be produced by an extended version of ODINS. In contrast to the unconstrained token passing search discussed in Section 3.2.2, in the constrained version only those search paths are taken into account which contain valid paths through the constraining lattice.

Constrained token passing is utilized for two tasks in this thesis: Firstly, to simulate a two-stage decoder that corresponds as much as possible to its one-stage counterpart. This enables to conduct an experimental comparison of the two approaches, which is presented in Section 3.3. Secondly, constrained token passing can be used to automatically generate semantic tree annotations from orthographic transcriptions of spoken utterances, as discussed in Section 4.1.2.

The constrained search is carried out by augmenting tokens with a reference to the previously traversed terminal node of the lattice. For the uses discussed in this thesis, terminal lattice nodes correspond to words. In this case, the knowledge source for the decoder is not a full WTNH including acoustic-phonetic and lexical modeling levels as in Chapter 2, but merely a WTNH comprising the syntactic-semantic modeling levels, just like the hierarchical language model presented in Chapter 5.

One measure to verify constraints consists in checking if tokens follow admissible transitions between terminal network nodes of WTNH. This is achieved by discarding tokens that visit a terminal network node if the node is, according to the lattice, not a possible successor of the previously traversed terminal network node. The

second action for constraint-verifying, which only needs to be carried out if the input lattice is hierarchical, consists in checking if the token's way through the non-terminal levels of the network hierarchy is admissible. This can be ensured by comparing hierarchy histories. More specifically, all tokens at terminal nodes are discarded whose super-network history is not contained in the possible super-lattice histories of the terminal lattice node. The super-lattice history is, similar to the super-network history, represented in the shape of a history tree (see Section 3.2.1). Hence, comparison of histories is performed by comparison of unordered (sub-)trees.

As mentioned in the beginning of this section, the input lattice can be weighted, for instance if it is the result of speech recognition. In this case, two different kinds of weights are present in the lattice, namely acoustic model and language model scores. If the constrained token passing should act as a second processing stage after speech recognition, it is desirable to reuse the acoustic model scores of the input lattice, but discard the language model scores and use the (hierarchical) language model scores of the WTNH instead. This method is similar to re-scoring known from multi-pass speech recognition, only that the second stage here introduces additional, semantic information. If the input lattice is unweighted, as is the case when generating semantic tree annotations from word sequences, only the weights present in the WTNH are considered during constrained token passing.

3.3 Experiment: One-Stage vs. Two-Stage Decoding

As mentioned in the introduction of this thesis, a one-stage decoding approach for speech interpretation is desirable because it obeys the well-known principle of applying all sources of knowledge as early as possible. In contrast, the speech recognizer of a two-stage speech understanding system computes the most likely word hypothesis without considering semantic knowledge from the second stage. This becomes relevant when speech interpretation hypotheses are not very likely if only acoustic-phonetic, lexical and syntactic modeling is taken into account, but likely regarding semantic modeling. If such a hypothesis is already discarded by the speech recognizer without having the chance to become the overall most likely hypothesis, this early decision may cause an error. On the contrary, ODINS prevents these kinds of errors by considering the available knowledge models simultaneously.

In general, it is also possible to circumvent this problem in a two-stage system, by generating alternative word or utterance hypotheses in the first stage and modifying the second-stage decoder accordingly. However, runtime requirements may impose strong limitations on the amount of hypotheses which can be accounted for in the second stage. Therefore, it is desirable to quantify the degradation of semantic accuracy when reducing the amount of alternative hypotheses. In this section, the results of a corresponding experiment are presented[1].

Figure 3.5 depicts the basic building blocks of the one-stage and the two-stage

[1] See [TFLR03a] for a similar experiment, which was conducted with an earlier version of the knowledge models and test corpora used here.

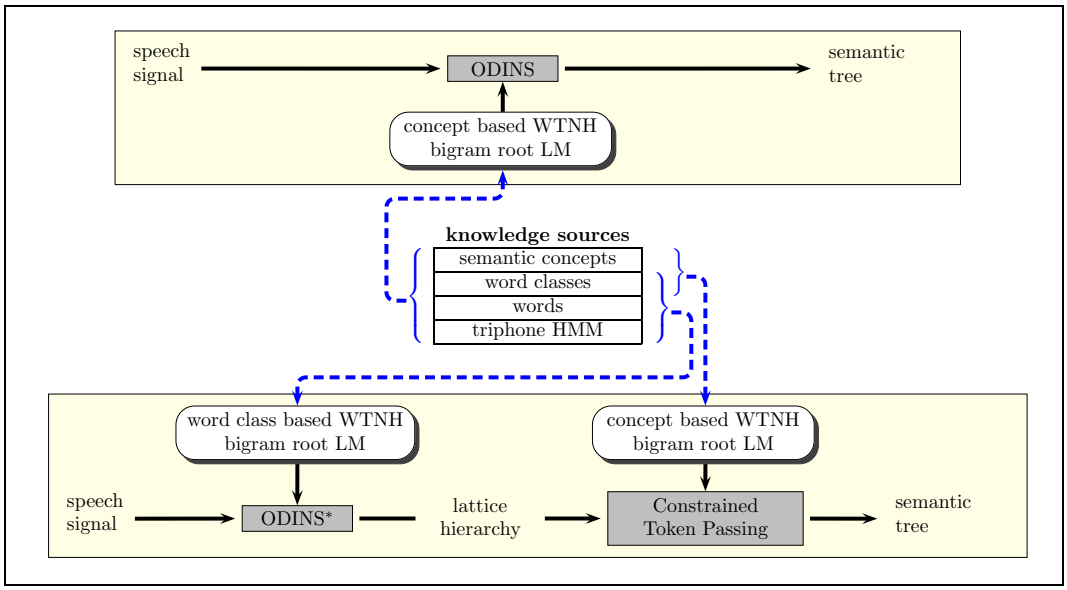


Figure 3.5: *Setup for comparing the one-stage (top) and the two-stage (bottom) decoders used in this thesis.*

decoders used in the experimental setup. In the one-stage system (see upper part of Figure 3.5), the WTNH incorporates all sources of knowledge we use for speech interpretation, namely acoustic-phonetic (HMM), lexical (word) and syntactic-semantic (word class and concept) models. In both two-stage and one-stage setups, acoustic-phonetic and lexical modeling is carried out as described in Section 5.6.2. The hierarchical language model of the WTNH for the one-stage setup is constructed as described in Chapter 5, using a training corpus of semantic tree annotations for an airport information dialogue scenario (see Section 4.1). Specifically, the transition network at the root of the network hierarchy represents a (concept) bigram language model. In the one-stage system, ODINS directly decodes semantic tree representations from input speech signals utilizing this WTNH.

For the first stage of the two-stage system setup (see lower part of Figure 3.5), a WTNH comprising only acoustic-phonetic, lexical and (implicit) syntactic knowledge is constructed using the same semantic tree annotations as for the one-stage setup, after discarding the concept annotation levels. Although word classes intrinsically belong to the semantic modeling levels, we incorporate them into the first decoding stage, because word class members required for the application but missing in the training corpus are added manually (see also Section 5.6.1). Therefore, omitting the word class models in the first stage would produce a smaller lexicon for the two-stage decoder, rendering the comparison unfair. Hence, the transition network at the root of the first-stage WTNH is based on word classes. As in the one-stage setup, a bigram is used as root language model. In order to produce alternative word and word class hypotheses, an extended version of ODINS (denoted as ODINS*) provides

the first-stage speech recognizer. This decoder generates so-called lattice hierarchies from speech signals (see Section 3.2.3).

This intermediate representation is further processed in the second stage of the two-stage speech interpretation system, utilizing the constrained token passing search discussed in Section 3.2.3. In order to decode semantic trees from lattice hierarchies of words and word classes, a second WTNH is used that contains the syntactic-semantic modeling levels. This WTNH is equivalent to the upper part of the WTNH for one-stage decoding, and also contains a concept bigram language model at the root level.

In the described experimental setup, both decoders correspond to each other as far as possible, in that they rely on the same training data, on the same modeling approach and on the same decoding scheme. Hence, this setup is suitable for measuring solely the effect of propagating only a limited number of hypotheses from the first to the second stage. The amount of alternatives present in a lattice hierarchy is expressed in terms of the lattice density ρ_{lat}, which is computed as the ratio of the total number of edges in the flat lattice N_e^{tot} and the number of nodes of the best path through the flat lattice N_n^{best}:

$$\rho_{lat} = \frac{N_e^{tot}}{N_n^{best}}$$

The lattice density value expresses how many alternative words and word classes a lattice contains on average for each word and word class in its best hypothesis. ρ_{lat} is controlled by varying two parameters of the first-stage decoder, namely the number of n-best tokens recorded at network nodes during lattice generation, and the beam pruning threshold t, which was discussed in Section 3.2.2.

For a set of $u = 1 \ldots U$ utterances from a test corpus, the average lattice density $\bar{\rho}_{lat}$ is computed as:

$$\bar{\rho}_{lat} = \frac{1}{U} \sum_{u=1}^{U} \rho_{lat}(u)$$

Figure 3.6 depicts the tree node accuracy Acc_n of the two-stage system setup in dependency of $\bar{\rho}_{lat}$ for the evaluation set of the airport information dialogue corpus. The tree node accuracy is a measure for the degree of agreement between a semantic reference tree and a semantic hypothesis tree, as discussed in Section 4.3. In other words, Acc_n specifies how much of the semantic tree structure was correctly interpreted.

The average tree node accuracy of the one-stage decoder is 87.5% for the evaluation set, shown as a constant line in Figure 3.6. This performance is achieved with a hierarchical language model utilizing Good-Turing and modified Kneser-Ney smoothing, as discussed in Section 5.7. For both one-stage and two-stage decoders the language model factor was set to $\lambda = 21$, which corresponds to the optimum setting on the cross-validation set (compare Figure 5.4).

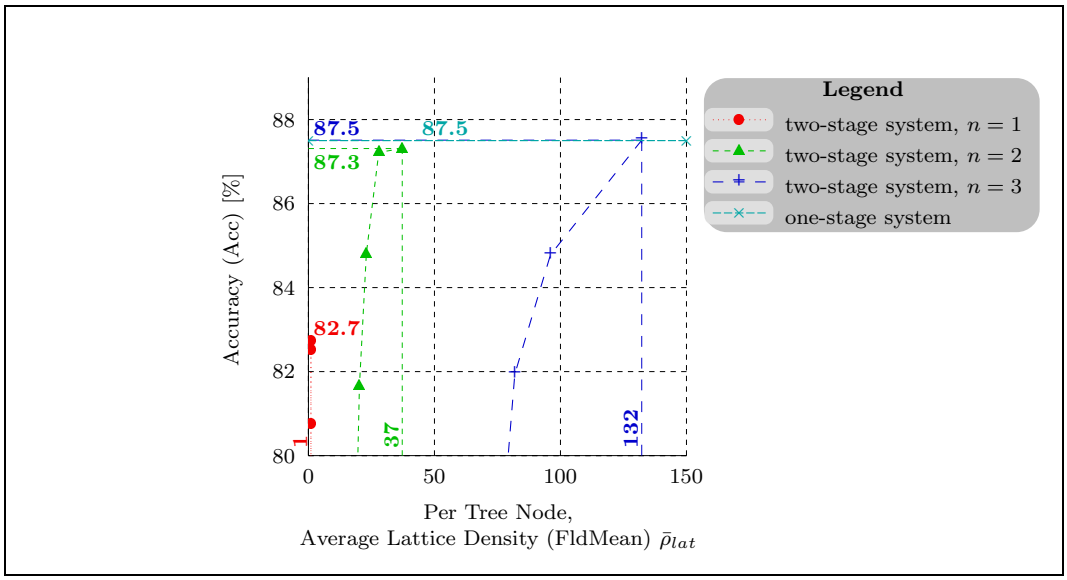


Figure 3.6: *Tree node accuracy of two-stage decoder for varying lattice densities.*

For the two-stage decoder three curves are drawn, corresponding to $n = 1$, $n = 2$ and $n = 3$ best tokens recorded during lattice generation. The beam pruning threshold was varied between $t = 80$ and $t = 400$. For $n = 1$ the lattice degenerates to the best search hypothesis, yielding a lattice density of $\bar{\rho}_{lat} = 1$ and a maximum tree node accuracy of only 82.7%. For $n = 2$ and an increasing amount of alternative hypotheses, the performance of the two-stage system continuously improves and achieves its maximum of $Acc_n = 87.3\%$ at $\bar{\rho}_{lat} = 37$, which comes close to the accuracy of the one-stage system of 87.5%. Further increasing the number of alternative hypotheses by setting $n = 3$ even yields the same accuracy as the one-stage system, yet at the cost of processing lattices with an average density of $\bar{\rho}_{lat} = 132$.

Summing up, the experiment suggests that a substantial amount of speech interpretation errors, which originate from decoding the semantic representation from a limited amount of speech recognizer hypotheses, can be avoided by a one-stage decoding approach. Using only the best hypothesis increases the error rate by more than 38% relative, compared to the error rate of the one-stage decoder. Although this error increase can be limited to about 2% or even eliminated altogether, lattices with more than 25 or even more than 100 alternatives for each word or word class in the best hypothesis are needed in order to achieve this. Processing such large lattices may well exceed the available computational resources as well as the requirements for runtime delay in a practical application.

Chapter 4

Evaluation Method and Measures

This chapter deals with the question how to evaluate a speech interpretation system, i.e. how to determine its 'goodness'. Answering this question is interesting for both manufacturers and customers of speech interpretation systems. Yet, finding a general measure of goodness is difficult, as the notion of goodness typically relates to a number of different system properties, which can be judged in different ways. Therefore, goodness also depends on the viewpoint of the observer.

This notion is especially relevant for complete, end-user systems such as dictation systems or spoken language dialogue systems, where personal customer preferences play an important role (see [BD00] for a more elaborate discussion on speech dialogue system goodness). But even single system components, such as the speech interpreter, may have a number of properties (e.g. word error rate, runtime performance, noise robustness) that can be judged and weighted differently. Hence, as also concluded by [BD00], evaluation should not aim to find 'the' goodness of a system. Instead, numerous factors contributing to system quality should be determined, which observers can consider according to their perspective, in order to make individual judgements of system goodness. We denote such a contributing factor as evaluation measure.

The provision of suitable evaluation measures is also important because it enables purposeful improvement of system components during system development. Moreover, comparison of different modeling and processing approaches or even of whole systems becomes possible.

Generally, evaluation measures can be divided into two categories, namely subjective, user centric measures and objective, system centric measures. System centric evaluation can usually be repeated with no or little additional cost once the evaluation environment has been created, whereas user centric evaluation generally consumes similar effort with every repetition of the test. Subjective evaluation is especially useful to determine quality aspects that the system developers are immune

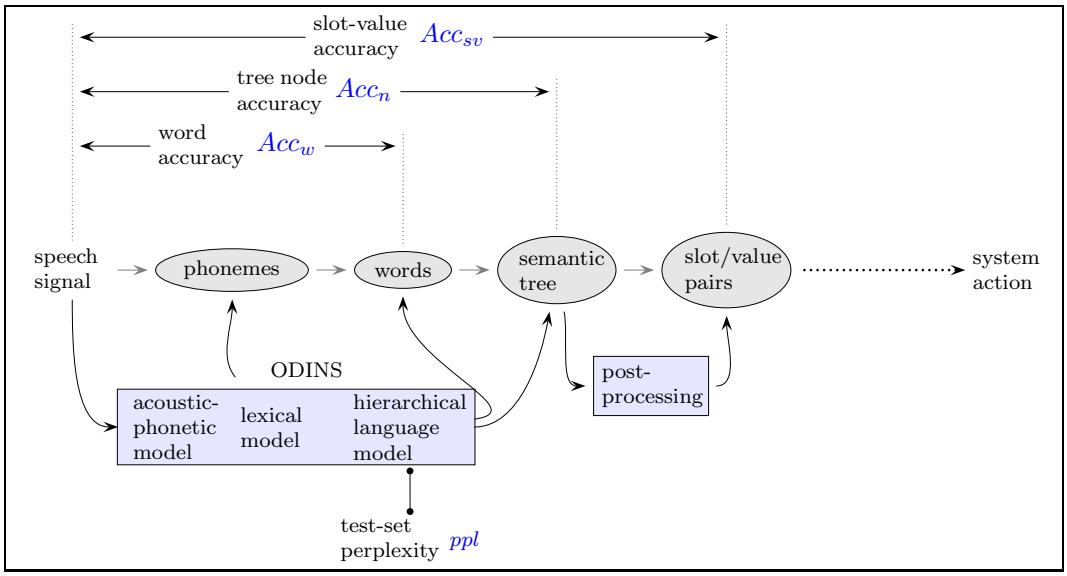


Figure 4.1: *Levels of representation and evaluation measures used for speech interpretation.*

against due to their involvement in the system design. Another possible categorization of evaluation methods is achieved by discriminating black box and glass box assessment [SF93]. The former denotes the evaluation of a system as a whole by only looking at its input and output, whereas the latter refers to the assessment of single system components only.

This thesis concentrates on objective, glass box evaluation of our speech interpretation system as a component of a spoken language dialogue system. The system centric evaluation is based on a speech corpus collected through a Wizard-of-Oz setup of an airport information dialogue system scenario. This speech corpus is discussed in Section 4.1. Yet, a complete speech dialogue system was not available for this thesis. Therefore, black box assessment as well as user centric evaluation couldn't be performed.

Figure 4.1 gives an overview over the evaluation measures considered in this chapter, namely word accuracy Acc_w, tree node accuracy Acc_n, slot-value accuracy Acc_{sv} and test-set perplexity ppl. In the figure, the scope of these measures is shown by means of the different levels of representation and processing: As explained previously, the one-stage decoder ODINS directly maps the input speech signal into a semantic tree representation. Internally, ODINS uses phonemes and words as (intermediate) representation, whereby words are also part of semantic trees. In a postprocessing step, a semantic tree is then converted into a suitable input representation for further processing, e.g. for the dialogue management component of a dialogue system. The postprocessing step e.g. involves the creation of a canonical

form of meaning (see Section 2.3.4). The input to the dialogue manager typically consists of slot-value pairs or slot-value-communicative function triples (see Section 4.2.3). Finally, the dialogue system executes an action, usually by responding to the user query.

Speech understanding systems are typically evaluated in terms of the goodness of the produced slot-value representation. Slot-value pairs are also called concepts in this context, and the corresponding evaluation measure denoted as concept accuracy. However, in this thesis the term slot-value accuracy is used instead, to avoid confusion with the semantic units of WTNH (see Section 4.2.3). In order to obtain evaluation data from single components of the speech understanding system, the goodness of the word representation, i.e. the speech recognition component, is also measured. Further insight into the speech recognition process is gained by evaluating the language model alone in terms of perplexity.

In this work, similar measures are defined to evaluate ODINS: Slot-value pairs are derived from semantic trees by postprocessing, word sequences are directly contained in the bottom level of semantic trees. Both evaluation measures are computed by sequence matching, which is discussed in Section 4.2. An assessment of the language modeling component alone is achieved by applying the notion of perplexity to the hierarchical language model of ODINS. The corresponding measure is defined in Section 4.4.

In addition to those metrics, a novel evaluation measure, which directly assesses the semantic tree representation, is introduced in this chapter: The so-called *tree node accuracy*, which is computed by tree matching instead of sequence matching. This measure is defined and discussed in Section 4.3, after reviewing the basic approach to tree matching and showing how it can be extended to match the semantic trees employed here. The section concludes with an experimental comparison between sequence based and tree based evaluation measures. The result suggests that tree based evaluation is better suited for comparably complex hierarchical models, like the ones examined in this work.

4.1 Airport Information Corpus

This section describes the collection and annotation of speech data for a sample application of a spoken language dialogue system, which contains the considered speech interpretation system as a component. One part of the annotated speech corpus provides the fundamental data for computation of the objective evaluation measures discussed in this chapter. Another part of the speech data is utilized for creation of knowledge sources of a sample speech interpretation system, especially for the hierarchical language models (HLM) discussed in Chapter 5.

The sample application proposes a dialogue system for information about the Munich airport. In the target scenario, the system user is driving by car to the airport in order to collect or deliver passengers, or to take a flight himself. The dia-

	lab	car	total
# subjects	15	17	32
# tasks/dialogues	240	255	495
# user utterances	1365	1323	2688
# words	7047	7624	14671

Table 4.1: *Statistics of speech data collected in laboratory and car environments.*

logue system provides the user with information about departing and arriving flights, specifically about departure and arrival times, about the origin or destination of a flight, about flight number, gate, status, and about the airplane type. Additionally, the user can request information about appropriate parking facilities.

4.1.1 Data Collection

The speech data was collected in a series of Wizard-of-Oz (WOZ) experiments, closer described in [LR03]. The WOZ technique [DJA93] is employed to obtain as realistic data as possible for a target domain without the need to provide a ready target system. 'Realistic' means that the collected utterances are similar to the ones expected in a real world dialogue application. In order to avoid degradation of speech interpretation accuracy, the similarity between collected and real-world utterances needs to extend over all levels of processing (acoustic, lexical, syntactic, semantic). This can be achieved if the subject is confronted with the a system that behaves real. WOZ experiments aim to meet this requirement by simulating essential parts of the dialogue system with the aid of a human 'wizard'. Ideally, subjects are unaware of the simulation, so that they behave more naturally. The wizard interprets the user utterances and initiates appropriate system actions, simulating speech interpretation and dialogue management.

The corpus collection was performed in two different environments: The first experiment took place in a quiet laboratory environment, the second one in a driving car, which the subject itself was steering. In both environments, the subjects were recorded with a close-talking microphone, in order to avoid influences of background noise. During the experiments, the subjects were given a number of tasks, each leading to a dialogue between the subject and the WOZ system. For each task, a short description containing target information (e.g. arrival time) and possible constraints (e.g. flight LH 4257 from frankfurt) was read to the subject by the experimenter. In some of the tasks, the wizard deliberately initiated erroneous system responses, simulating misinterpretation.

32 recordings of 30 different subjects were carried out in total. 15 different subjects were recorded in the laboratory experiment, 17 different subjects in the car, i.e. 2 subjects carried out both experiments. The number of tasks each subject had to perform was 16 in the laboratory and 15 in the car experiment. The resulting 495 dialogues consist of 2688 user utterances or 14671 words, so that on average a

dialogue contains 5.4 utterances, and an utterance contains 5.5 words. Table 4.1 summarizes the basic properties of the two data collections.

4.1.2 Corpus Annotation and Partitioning

The collected utterances from the airport information scenario are fully annotated with semantic trees (see Section 2.3.3). In order to reduce the manual effort required for annotation, a semi-automatic, iterative procedure is employed: Firstly, an initial HLM is built from a small amount of manual tree annotations. Then, tree annotations of new utterances are generated automatically from the word sequence, using the constrained token passing scheme presented in Section 3.2.3. After manually correcting the errors in the automatic annotations, a new intermediate HLM is built on the extended set of annotated utterances. This process is repeated iteratively for chunks of the corpus, as the tagging accuracy improves with the incorporation of new phenomena into the intermediate HLM.

All of the objective evaluation measures mentioned in the beginning of this chapter are computed on a set of test utterances of the considered speech corpus. Furthermore, the speech corpus is also utilized to train the knowledge sources of the speech interpretation system (HMM, lexicon and HLM). Hence, in order to render the evaluation as fair and as practical as possible, the corpus needs to be split into training and test sets so that the test set contains no data that is already known to the system through training. This especially requires that the test set only contains utterances from previously unseen subjects.

The test set is again split into two parts, namely into an evaluation set and a cross-validation set. The cross-validation set is used to find the optimum settings of free parameters of the speech interpretation system, such as language model factor or OOV word penalties (see Sections 5.9.1 and 6.3). With those parameter settings, the final evaluation result is then determined on the evaluation set, without using knowledge about the optimum parameter settings on this set.

The tree annotation is based on a definition of 53 semantic categories. 12 of those categories are defined on the word class level, the other 41 categories represent semantic concepts, arranged on two hierarchy levels. The structural distinction between word classes and semantic concepts is, that a word class represents a union of single words, whereas a concept may contain several subordinate units, either terminal (words) or non-terminal (other concepts or word classes). In semantic terms, members of the word class category always relate to information items the dialogue system can give advice about (e.g. airport locations, airline codes, area codes, digits of flight numbers), whereas concepts also represent other kinds of relations (see Section 2.3.4). Some statistical data on the annotated airport information corpus is given in Table 4.2. The table contains the total numbers for the whole corpus (all) as well as individual numbers for training (train), evaluation (eval) and cross-validation (xval) sets.

	airport information corpus			
	train	eval	xval	all
# subjects	20	6	6	32
# dialogues	309	93	93	495
# user utterances	1714	439	535	2688
# words	9616	2496	2559	14671
# words per utterance	5.6	5.8	4.7	5.5
# word classes	3689	1036	1019	5744
# word classes per utt.	2.2	2.4	1.9	2.1
# concepts	3904	1018	1098	6020
# concepts per utterance	2.3	2.3	2.1	2.2
# different words	594	307	272	667
# different word classes	12	8	12	12
# different concepts	41	41	38	41
unknown word rate	n.a.	2.2%	1.5%	n.a.

Table 4.2: *Statistics of semantic tree annotation and partitioning into training and test sets.*

4.2 Sequence Based Evaluation

After discussing collection and annotation of a speech corpus for a sample dialogue scenario and its partitioning into training and test sets, this section now deals with sequence matching based evaluation of speech interpretation systems. For this purpose, annotations of test utterances are utilized as reference material. Evaluation measures are then computed by comparing these references against the output hypotheses of the speech interpretation decoder for the recorded speech signals of those test utterances. Sequence based evaluation focuses on matching sequential representations of utterance hypotheses and references. We specifically review two objective evaluation measures typically used for speech understanding systems, namely word accuracy and slot-value accuracy. As shown in Figure 4.1 in the beginning of this chapter, these metrics consider different levels of representation, and therefore different processing stages.

4.2.1 Sequence Matching

In order to compute a sequence based evaluation measure, the optimum match between reference and hypothesis sequences needs to be found, so that the erroneous elements of the sequences can be located. The optimum match between two sequences is usually determined as the minimum edit distance, also called Levenshtein distance, whereby this distance is computed as the sum of the costs of a set of edit operations required to transform one sequence into another. By definition, there are three different edit operations on sequence elements, namely insertions, deletions

and substitutions. An insert operation inserts a new element into the sequence, a delete operation deletes an element, and a substitution replaces an element by a different one. Elements that don't need changing are denoted as correct.

Formally, such a transformation between two sequences $\mathbf{S}_1$ and $\mathbf{S}_2$ can be viewed as an unordered set of $|M|$ edit operations, denoted as *mapping* M:

$$M(\mathbf{S}_1, \mathbf{S}_2) = \{(i_1 \mapsto j_1), (i_2 \mapsto j_2), .., (i_{|M|} \mapsto j_{|M|})\}$$

An edit operation is written as $(i \mapsto j)$, where i and j are indices $1 \leq i \leq |\mathbf{S}_1|$ or $1 \leq j \leq |\mathbf{S}_2|$ from the symbol sequences $\mathbf{S}_1 = [\mathbf{S}_1[1], \mathbf{S}_1[2], .., \mathbf{S}_1[|\mathbf{S}_1|]]$ or $\mathbf{S}_2 = [\mathbf{S}_2[1], \mathbf{S}_2[2], .., \mathbf{S}_2[|\mathbf{S}_2|]$, respectively, or the null index ε. The edit operation $(i \mapsto j)$ is called a *map operation* if $i \neq \varepsilon$ and $j \neq \varepsilon$, a *delete operation* if $i \neq \varepsilon$ and $j = \varepsilon$ and an *insert operation* if $i = \varepsilon$ and $j \neq \varepsilon$. A map operation $(i \mapsto j)$ is termed *correct* if the symbols are identical ($\mathbf{S}_1[i] = \mathbf{S}_2[j]$), or a *substitution* otherwise ($\mathbf{S}_1[i] \neq \mathbf{S}_2[j]$). The ith symbol of a sequence $\mathbf{S}$ of length $|\mathbf{S}|$ is denoted $\mathbf{S}[i]$. A sub-sequence of $\mathbf{S}$ that contains consecutive elements from index i' to index i inclusively is denoted $\mathbf{S}[i'..i]$. Moreover, $\mathbf{S}[i..i] = \mathbf{S}[i]$ and $\mathbf{S}[i'..i] = \emptyset$ if $i < i'$, where $\emptyset$ denotes an empty sequence. By definition, any pair of map operations $(i_1 \mapsto j_1)$ and $(i_2 \mapsto j_2)$ with $1 \leq i_1, i_2 \leq |\mathbf{S}_1|$ and $1 \leq j_1, j_2 \leq |\mathbf{S}_2|$ of a mapping $M(\mathbf{S}_1, \mathbf{S}_2)$ has the following properties:

$$\begin{array}{lcll} i_1 = i_2 & \iff & j_1 = j_2 & \text{(one-to-one mapping)} \\ i_1 < i_2 & \iff & j_1 < j_2 & \text{(symbol order preserved)} \end{array} \tag{4.1}$$

The distance of a mapping M is measured by introducing a cost function $c(i \mapsto j)$ for each edit operation $(i \mapsto j)$. In order to be a distance metric, c has to meet the following conditions:

$$\begin{array}{lcll} c(x \mapsto y) & \geq & 0 & \text{(non-negative cost)} \\ c(x \mapsto x) & = & 0 & \text{(no cost for identity mapping)} \\ c(x \mapsto y) & = & c(y \mapsto x) & \text{(order independent)} \\ c(x \mapsto z) & \leq & c(x \mapsto y) + c(y \mapsto z) & \text{(triangle inequality)} \end{array} \tag{4.2}$$

In this thesis, the cost function is defined as in the widely used NIST scoring software [FF93] as:

$$c(i \mapsto j) = \begin{cases} 0, & i, j \neq \varepsilon \text{ and } \mathbf{S}_1[i] = \mathbf{S}_2[j] \quad \text{(correct)} \\ 4, & i, j \neq \varepsilon \text{ and } \mathbf{S}_1[i] \neq \mathbf{S}_2[j] \quad \text{(substitution)} \\ 3, & i \neq \varepsilon \text{ and } j = \varepsilon \quad \text{(deletion)} \\ 3, & i = \varepsilon \text{ and } j \neq \varepsilon \quad \text{(insertion)} \end{cases} \tag{4.3}$$

The edit distance $D_e(M(\mathbf{S}_1, \mathbf{S}_2))$ of a mapping $M(\mathbf{S}_1, \mathbf{S}_2)$ is then computed by accumulating the costs of the single edit operations:

$$D_e(M(\mathbf{S}_1, \mathbf{S}_2)) = \sum_{(i,j) \in M(\mathbf{S}_1, \mathbf{S}_2)} c(i \mapsto j)$$

$j\downarrow$	$i\rightarrow$	0	1 LH	2 EINS	3 NULL	4 NEUN	5 NACH	6 HAMBURG
0		*0*	→ *3*	→ 6	→ 9	→ 12	→ 15	→ 18
		↓ ↘	↘		↘			
1	EINS	3	4	*3*	→ 6	→ 9	→ 12	→ 15
		↓ ↘	↓	↓ ↘	↘	↘	↘	
2	ZWEI	6	7	*6*	7	→ 10	→ 13	→ 16
		↓ ↘	↓	↓ ↘				
3	NULL	9	10	9	*6*	→ 9	→ 12	→ 15
		↓ ↘	↓ ↘		↓ ↘			
4	NEUN	12	13	10	9	*6*	→ 9	→ 12
		↓ ↘	↓	↓	↓	↓ ↘		
5	VON	15	16	13	12	9	*10*	→ 13
		↓ ↘	↓	↓	↓	↓ ↘	↓ ↘	
6	HAMBURG	18	19	16	15	12	13	*10*

Figure 4.2: *Example for word sequence matching by dynamic programming.*

Hence, the minimum edit distance $D_e^{min}(\mathbf{S}_1, \mathbf{S}_2)$ is defined as the edit distance of the specific mapping that yields the minimum distance value:

$$D_e^{min}(\mathbf{S}_1, \mathbf{S}_2) = \min_{M(\mathbf{S}_1, \mathbf{S}_2)} D_e(M(\mathbf{S}_1, \mathbf{S}_2)) \tag{4.4}$$

The task of solving Equation 4.4 for given sequences $\mathbf{S}_1$ and $\mathbf{S}_2$ and a given cost function c is usually performed by a dynamic programming (DP) algorithm [SK83, Rus94]. With the DP algorithm, the minimum edit distance is computed by accumulating the single costs c step-by-step in a left-to-right manner. Locally, i.e. in each step, decisions are taken to yield the least-cost path. Thus, the minimum edit distance is determined with a recursive formula that computes the accumulated cost for transforming the sub-sequence $\mathbf{S}_1[1..i]$ into $\mathbf{S}_2[1..j]$ from previously computed values:

$$D_e^{min}(\mathbf{S}_1[1..i], \mathbf{S}_2[1..j]) = \min \begin{cases} D_e^{min}(\mathbf{S}_1[1..i-1], \mathbf{S}_2[1..j]) & + c(i \mapsto \varepsilon) \\ D_e^{min}(\mathbf{S}_1[1..i], \mathbf{S}_2[1..j-1]) & + c(\varepsilon \mapsto j) \\ D_e^{min}(\mathbf{S}_1[1..i-1], \mathbf{S}_2[1..j-1]) & + c(i \mapsto j) \end{cases} \tag{4.5}$$

Figure 4.2 illustrates the DP based minimum edit distance computation for a reference word sequence $\mathbf{S}_1$ = [LH, EINS, NULL, NEUN, NACH, HAMBURG] and a hypothesis word sequence $\mathbf{S}_2$ = [EINS, ZWEI, NULL, NEUN, VON, HAMBURG]. The minimum edit distances $D_e^{min}(\mathbf{S}_1[1..i], \mathbf{S}_2[1..j])$ are depicted as cells of a table whose columns and rows correspond to the indices of $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. The table is processed top-to-bottom and left-to-right. According to Equation 4.5, the value of a

i	1	2		3	4	5	6
$\mathbf{S}_1[i]$	LH	EINS		NULL	NEUN	NACH	HAMBURG
		↓		↓	↓	↓	↓
$\mathbf{S}_2[j]$		EINS	ZWEI	NULL	NEUN	VON	HAMBURG
j		1	2	3	4	5	6

Figure 4.3: *Minimum edit distance mapping for example of Figure 4.2.*

cell (i, j) is computed from the cells to the left $(i-1, j)$, to the top $(i, j-1)$, and to the top-left $(i-1, j-1)$. Transitions from the left are computed with the first term of Equation 4.5 and correspond to delete operations; ones from the top correspond to the second term and to insertions; ones from the top-left correspond to the third term and to correct matches if $\mathbf{S}_1[i] = \mathbf{S}_2[j]$ or to substitutions otherwise. The best transition corresponds to the minimum of the three values and is depicted by an arrow.

Generally, more than one transition can result in the same optimum value, e.g. at cell $(1, 2)$ both the transitions from the top and from the top-left yield the minimal cost. Hence, a pair of sequences can have more than one minimum edit distance mapping. The example of Figure 4.2 has only one optimum mapping with an edit distance of 10 (the value of the cell $(|\mathbf{S}_1|, |\mathbf{S}_2|)$). The optimum mappings are retrieved by tracking the transitions from cell $(|\mathbf{S}_1|, |\mathbf{S}_2|)$ back to cell $(0, 0)$, depicted in Figure 4.2 with italicized numbers. In order to allow delete and insert operations at the beginning of the sequences, the table contains an extra column and row corresponding to empty sequences $\emptyset$. The table is initialized by setting the value of cell $(0, 0)$ to $D_e^{min}(\emptyset, \emptyset) = 0$.

Figure 4.3 depicts the optimum mapping resulting from the example of Figure 4.2. The mapping contains three erroneous elements, namely a deletion $(1 \mapsto \varepsilon)$, an insertion $(\varepsilon \mapsto 2)$ and a substitution $(5 \mapsto 5)$. Hence, the edit distance for this mapping is $D_e(M(\mathbf{S}_1, \mathbf{S}_2)) = 10$.

4.2.2 Word Accuracy

A standard evaluation measure for the assessment of speech recognition systems is word accuracy. The word accuracy is determined by matching reference word sequence against hypothesis word sequence for a set of test utterances, as shown in the example of Figure 4.3. Let C_w, S_w, I_w and D_w denote the total number of correct, substituted, inserted and deleted words over a set of (optimum) mappings between word sequences. Then, the word accuracy Acc_w is computed by subtracting the number of erroneous map operations (substitutions, insertions and deletions) from the total number of words in the references $N_w = C_w + S_w + D_w$, and normalizing appropriately:

$$Acc_w = \frac{N_w - (S_w + I_w + D_w)}{N_w} = 1 - \frac{S_w + I_w + D_w}{N_w} \tag{4.6}$$

Instead of reporting results as accuracy values, error rates can be used instead. For this thesis, we generally define an error rate Err in terms of an accuracy Acc as:

$$Err = 1 - Acc$$

This definition not only applies to word accuracy, but also to the other accuracy measures defined in following sections.

Sometimes, word insertions are not considered as errors, especially when evaluating speech recognition tasks. The evaluation measure is then called word correctness. For speech interpretation tasks, however, where insertions of words or semantic units could cause wrong system actions, we view insertions as errors. Consequently, the word accuracy values are in the range $-\infty < Acc_w \leq 100\%$. The example mapping of Figure 4.3 yields a words accuracy of $Acc_w = 1 - \frac{1+1+1}{6} = 50\%$, if it is assumed that the reference sequence is represented by $\mathbf{S}_1$ and the hypothesis by $\mathbf{S}_2$.

If the considered hypotheses are the output of a speech recognition system, word accuracy is a black box measure of the overall system goodness considering all involved knowledge sources. Yet, in the one-stage speech interpretation system setup examined in this thesis, word accuracy measurements are also influenced by the semantic knowledge contained in the HLM (compare Figure 4.1). As already mentioned in the introduction, the ultimate goal of this system is not a perfect word transcription, but the perfect semantic tree representation. Therefore, optimization of system parameters is rather guided by evaluation measures that also take semantic modeling performance into account, and not by maximizing word accuracy, as is often the case in speech recognition. This needs to be taken into consideration when judging the significance of word accuracy in the context of one-stage speech interpretation (see Section 5.10).

4.2.3 Slot-Value Accuracy

Speech understanding systems typically produce meaning representations in the shape of slot-value pairs [BEG+96, Min98, SF93] or slot-value-communicative function triples [Bod98, vBKN99]. Therefore, objective black box assessment of speech understanding systems can be achieved by matching sequences of slot-value pairs. Boros et al. [BEG+96] define such an evaluation measure in a similar way as word accuracy, and call this concept accuracy. In order to avoid confusion with our notation of semantic units (compare Figure 2.2), we use the term slot-value accuracy instead.

In contrast to word accuracy, now the basic elements of reference and hypothesis sequences are no single units, but pairs. Hence, we modify the basic definition of a sequence $\mathbf{S}$ from Section 4.2.1, so that each element i of $\mathbf{S}$ now contains two symbols, namely a slot symbol $\mathbf{S}^{slot}[i]$ and a value symbol $\mathbf{S}^{val}[i]$. Consequently, the notion of mappings needs to adapted as well. In accordance with [BEG+96], we define edit

operations between sequences of slot-value pairs as:

$$(i \mapsto j) \text{ is } \begin{cases} \text{correct} & \iff i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] = \mathbf{S}_2^{slot}[j] \text{ and } \mathbf{S}_1^{val}[i] = \mathbf{S}_2^{val}[j] \\ \text{substitution} & \iff i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] = \mathbf{S}_2^{slot}[j] \text{ and } \mathbf{S}_1^{val}[i] \neq \mathbf{S}_2^{val}[j] \\ \text{deletion} & \iff (i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] \neq \mathbf{S}_2^{slot}[j]) \text{ or } (i \neq \varepsilon \text{ and } j = \varepsilon) \\ \text{insertion} & \iff (i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] \neq \mathbf{S}_2^{slot}[j]) \text{ or } (i = \varepsilon \text{ and } j \neq \varepsilon) \end{cases}$$

Hence, a map operation between an element i of sequence $\mathbf{S}_1$ and an element j of sequence $\mathbf{S}_2$ is only correct if both the slot symbols and the value symbols match. If the slots match but the values don't, the map operation is defined as substitution. If the slots don't match, the edit operation is not treated as a mapping at all, but as an insert and a delete operation.

This definition of edit operations can be implemented by redefining the cost function of Equation 4.3 as:

$$c(i \mapsto j) = \begin{cases} 0, & i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] = \mathbf{S}_2^{slot}[j] \text{ and } \mathbf{S}_1^{val}[i] = \mathbf{S}_2^{val}[j] & \text{(correct)} \\ 4, & i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] = \mathbf{S}_2^{slot}[j] \text{ and } \mathbf{S}_1^{val}[i] \neq \mathbf{S}_2^{val}[j] & \text{(subst.)} \\ 3, & i \neq \varepsilon \text{ and } j = \varepsilon & \text{(deletion)} \\ 3, & i = \varepsilon \text{ and } j \neq \varepsilon & \text{(insertion)} \\ \infty, & i,j \neq \varepsilon \text{ and } \mathbf{S}_1^{slot}[i] \neq \mathbf{S}_2^{slot}[j] & \text{(del./ins.)} \end{cases}$$

Using this definition, the standard minimum edit distance computation presented in Section 4.2.1 can be applied.

Let C_{sv}, S_{sv}, I_{sv} and D_{sv} denote the total number of correct, substituted, inserted and deleted slot-value pairs over a set of (optimum) mappings between slot-value pair sequences. Then, the slot-value accuracy Acc_{sv} is defined similar to word accuracy (compare Equation 4.6) as

$$Acc_{sv} = 1 - \frac{S_{sv} + I_{sv} + D_{sv}}{N_{sv}} \tag{4.7}$$

where $N_{sv} = C_{sv} + S_{sv} + D_{sv}$ denotes the total number of slot-value pairs in the reference sequences.

4.3 Tree Based Evaluation

In the previous sections, two standard measures used for speech recognition and speech understanding system evaluation were discussed, word accuracy and slot-value accuracy. Yet, as shown in Figure 4.1, for assessment of the one-stage speech interpretation system regarded in this work, a novel, further objective evaluation measure is employed. This metric is directly determined from the semantic tree representation produced by ODINS, and denoted as tree node accuracy.

The computation of this tree based evaluation measure follows the known scheme of comparing reference annotations of a set of test utterances with the output hypothesis of the decoder for the recorded speech signals of those test utterances. Here, the references are the semantic tree annotations discussed in Section 4.1.2, and the

decoder hypothesis are the semantic trees produced by ODINS. For matching this kind of tree structures, a number of algorithms have been proposed (see Shasha and Zhang [SZ97] for an overview). Those algorithms have been applied to a variety of tasks, e.g. RNA secondary structure analysis [SZ90]. In this work, we propose a novel use of this principle, which enables a direct comparison of the considered semantic trees.

For this task, we utilize an algorithm by Shasha and Zhang which is general, straight-forward to understand and to implement and has limited time and space complexity. Its basic scheme is outlined in Section 4.3.1. Section 4.3.2 then discusses how this principle can be adapted to compare the specific tree representation used in this thesis, and presents a definition of semantic tree node accuracy. This definition was also published in [TFLR03b]. An experimental comparison between this novel metric and standard sequence based evaluation is given in Section 4.3.3. The experiment shows that tree based evaluation is better suited for the kind of speech interpretation system examined in this work.

4.3.1 Tree Matching

In this section, we outline a matching technique for ordered, labeled trees, which was originally proposed by Shasha and Zhang. For a more detailed discussion the interested reader is referred to [SZ97].

By definition, a labeled, ordered, rooted tree $\mathbf{T}$ consists of $|\mathbf{T}|$ tree nodes, exactly one of them being the root node. A tree node of $\mathbf{T}$ is referred to via its index i; the symbol or label attached to the node with index i is denoted $t[i]$. As in [SZ97], we assume in the following that tree node indices are assigned to $\mathbf{T}$ by left-to-right postorder numbering. This numbering scheme is beneficial for a recursive formulation of the tree matching algorithm, as discussed later in this section. Left-to-right postorder numbers can be computed by performing a left-right depth search through $\mathbf{T}$, assigning each node a consecutive number after all of its children have been visited. Hence, $i = 1$ refers to the leftmost leaf node, and $i = |\mathbf{T}|$ refers to the root node of $\mathbf{T}$. Figure 4.4 illustrates two example trees $\mathbf{T}_1$ and $\mathbf{T}_2$ along with their postorder indices, shown in brackets below the node symbol.

Using these basic notions, the problem of finding the optimum match between two ordered, labeled trees $\mathbf{T}_1$ and $\mathbf{T}_2$ is approached, similar to sequence matching, by determining a minimum edit distance transformation from $\mathbf{T}_1$ to $\mathbf{T}_2$. For this purpose, a mapping M between two labeled, ordered trees $\mathbf{T}_1$ and $\mathbf{T}_2$ is defined as an unordered set of tree edit operations $(i \mapsto j)$:

$$M(\mathbf{T}_1, \mathbf{T}_2) = \{(i_1 \mapsto j_1), (i_2 \mapsto j_2), .., (i_{|M|} \mapsto j_{|M|})\}$$

The indices $i_1, i_2, .., i_{|M|}$ and $j_1, j_2, .., j_{|M|}$ are either tree node indices in the ranges $1..|\mathbf{T}_1|$ or $1..|\mathbf{T}_2|$, respectively, or the null index ε. The edit operation $(i \mapsto j)$ is called a *map operation* if $i \neq \varepsilon$ and $j \neq \varepsilon$, a *delete operation* if $i \neq \varepsilon$ and $j = \varepsilon$ and an *insert operation* if $i = \varepsilon$ and $j \neq \varepsilon$.

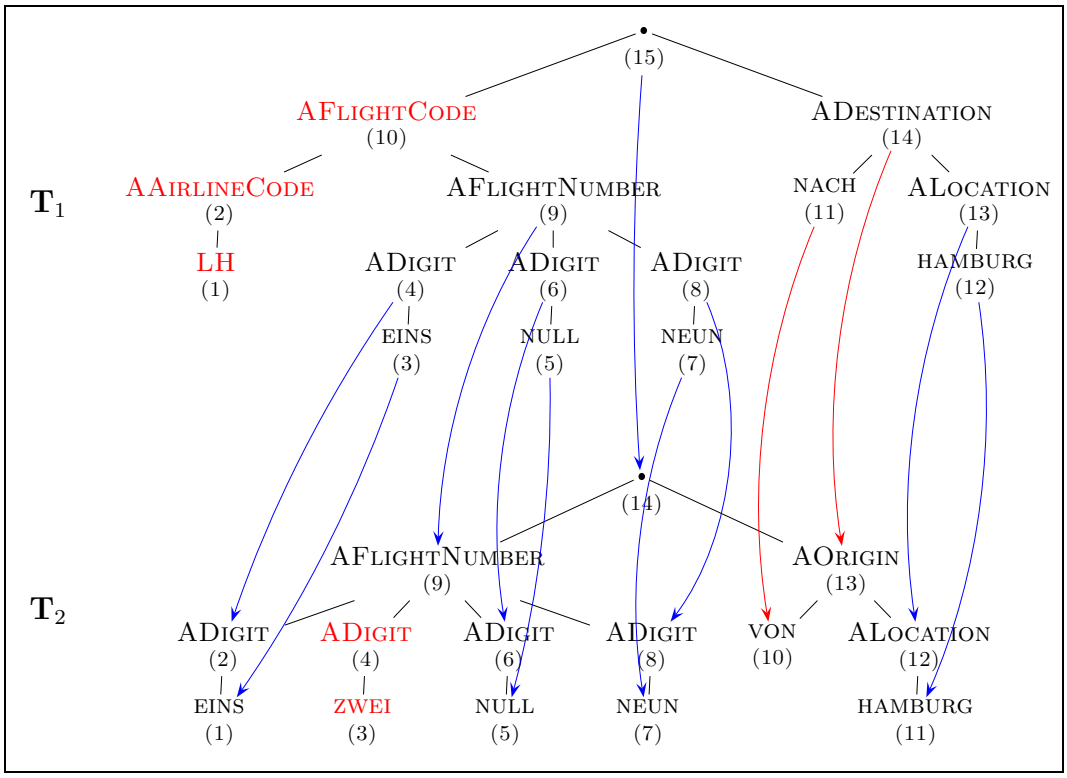


Figure 4.4: *Example mapping between two labeled, ordered, rooted trees* $\mathbf{T}_1$ *and* $\mathbf{T}_2$*. Node indices (in brackets) are assigned by left-to-right postorder numbering.*

Edit operations on tree nodes are defined as illustrated in Figure 4.5: A map operation converts a tree node i into a node j; a deletion removes a node i and reassigns its children to its parent k; a node j is inserted as child of a node k so that a consecutive number of k's children become the children of j.

An example mapping between two trees is shown in Figure 4.4, where map operations are illustrated by arrows between pairs of tree nodes. In this example mapping, nodes 1, 2 and 10 are deleted from $\mathbf{T}_1$, nodes 3 and 4 are inserted into $\mathbf{T}_2$, and the symbols of nodes 11 and 14 of $\mathbf{T}_1$ are substituted. This is formally written as:

$$\begin{aligned} M(\mathbf{T}_1, \mathbf{T}_2) \quad = \quad & \{(1,\varepsilon),(2,\varepsilon),(3,1),(4,2),(\varepsilon,3),(\varepsilon,4),(5,5),(6,6),(7,7), \\ & (8,8),(9,9),(10,\varepsilon),(11,10),(12,11),(13,12),(14,13),(15,14)\} \end{aligned}$$

Generally, any pair of map operations $(i_1 \mapsto j_1)$ and $(i_2 \mapsto j_2)$ with $1 \leq i_1, i_2 \leq |\mathbf{T}_1|$ and $1 \leq j_1, j_2 \leq |\mathbf{T}_2|$ of a mapping $M(\mathbf{T}_1, \mathbf{T}_2)$ must meet the following conditions:

$$\begin{aligned} i_1 = i_2 &\iff j_1 = j_2 && \text{(one-to-one mapping)} \\ i_1 \text{ left sibling of } i_2 &\iff j_1 \text{ left sibling of } j_2 && \text{(sibling order preserved)} \\ i_1 \text{ ancestor of } i_2 &\iff j_1 \text{ ancestor of } j_2 && \text{(ancestor order preserved)} \end{aligned} \tag{4.8}$$

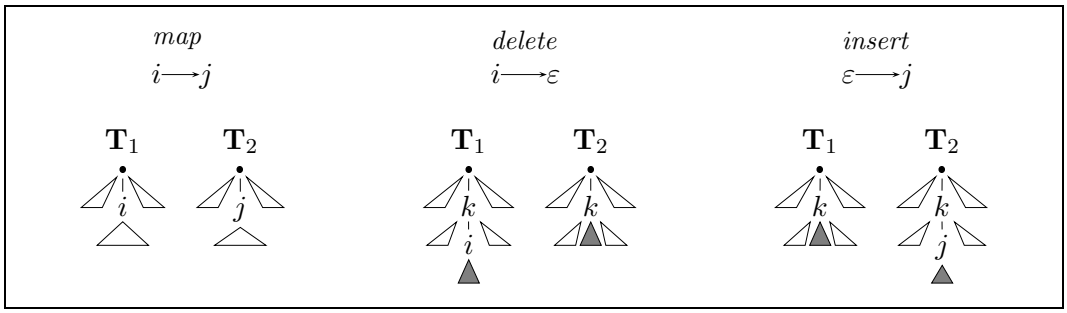


Figure 4.5: *Edit operations on tree nodes; a triangle symbolizes a sub-tree of the node at its top.*

Similar to the mapping conditions for the sequence matching case of Equation 4.1, the first two conditions of Equation 4.8 state that a tree node may not occur in more than one map operation, and that the transformation preserves the left-to-right order of a tree. Additionally, the third condition of Equation 4.8 ensures preservation of the bottom-to-top (vertical) tree structure. The latter e.g. means that if $t_1[14] =$ ADESTINATION is mapped to $t_2[13] =$ AORIGIN in Figure 4.4, a descendant of ADESTINATION such as NACH may only be mapped to a node 'below' AORIGIN, more precisely to a node contained in the sub-tree rooted at AORIGIN.

Again, the best match is determined through the mapping yielding minimum edit distance. Therefore, mappings are associated with distances by assigning a cost to each tree edit operation and accumulating the single costs. As for edit operations on sequences, the cost function c for tree edit operations $(i \mapsto j)$ needs to fulfill Conditions 4.2 in order to be a distance metric. Hence, a valid definition of c in accordance with [FF93] is, similar to Equation 4.3:

$$c(i \mapsto j) = \begin{cases} 0, & i, j \neq \varepsilon \text{ and } t_1[i] = t_2[j] \quad \text{(correct)} \\ 4, & i, j \neq \varepsilon \text{ and } t_1[i] \neq t_2[j] \quad \text{(substitution)} \\ 3, & i \neq \varepsilon \text{ and } j = \varepsilon \quad \text{(deletion)} \\ 3, & i = \varepsilon \text{ and } j \neq \varepsilon \quad \text{(insertion)} \end{cases} \tag{4.9}$$

Finally, the minimum tree edit distance $D_t^{min}(\mathbf{T}_1, \mathbf{T}_2)$ of the mapping $M(\mathbf{T}_1, \mathbf{T}_2)$ that transforms $\mathbf{T}_1$ into $\mathbf{T}_2$ is expressed by:

$$D_t^{min}(\mathbf{T}_1, \mathbf{T}_2) = \min_{M(\mathbf{T}_1, \mathbf{T}_2)} \sum_{(i,j) \in M(\mathbf{T}_1, \mathbf{T}_2)} c(i \mapsto j)$$

The determination of $D_t^{min}(\mathbf{T}_1, \mathbf{T}_2)$ can be carried out with a recursive, DP-style computation scheme. Its basic principle consists of computing the accumulated costs for transforming sub-structures from a tree $\mathbf{T}_1$ to sub-structures from a tree $\mathbf{T}_2$ in a step-by-step, left-to-right manner.

For this purpose, the postorder numbering is used to outline sub-structures of an ordered tree $\mathbf{T}$, which consist of consecutively numbered tree nodes. Generally,

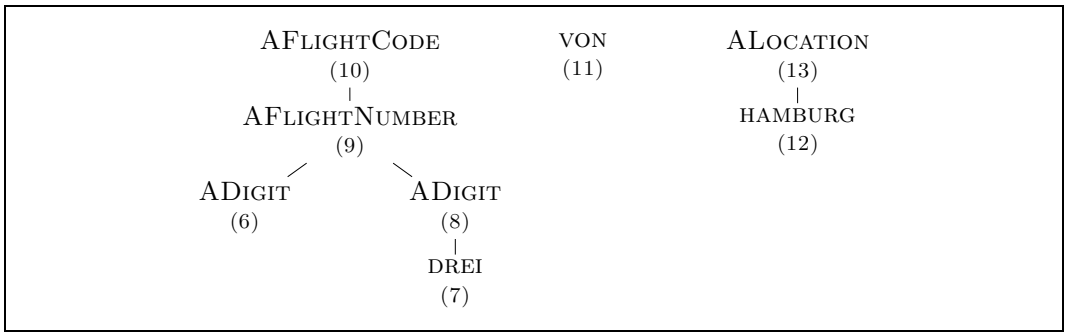


Figure 4.6: *Ordered sub-forest* $\mathbf{T}_1[6..13]$ *of tree* $\mathbf{T}_1$ *of Figure 4.4.*

such a structure does not consist of a single sub-tree of $\mathbf{T}$ but of several sub-trees, denoted as ordered sub-forest of $\mathbf{T}$. The sub-forest is ordered in the sense that its sub-trees appear in the same order as they do in $\mathbf{T}$. Formally, $\mathbf{T}[i'..i]$ denotes the sub-forest consisting of the sub-trees of $\mathbf{T}$ that contain the nodes with indices i' to i, inclusively. If $i' > i$, $\mathbf{T}[i'..i] = \emptyset$. Figure 4.6 depicts an example sub-forest of tree $\mathbf{T}_1$ of Figure 4.4, consisting of nodes 6..13 of $\mathbf{T}_1$.

Consequently, in order to compute distances between trees, distances between forests need to be determined as an intermediate step. More specifically, the minimum forest distance $D_f^{min}(\mathbf{T}_1[i'..i], \mathbf{T}_2[j'..j])$ or shorter $D_f^{min}(i'..i, j'..j)$ between the sub-forests $\mathbf{T}_1[i'..i]$ and $\mathbf{T}_2[j'..j]$ is required. Then, a recursive formulation similar to sequence matching (see Equation 4.5) can be devised, which determines minimum forest distances from previously computed distances and the costs for the three tree edit operations.

For delete and insert operations, the computation is equivalent to sequence matching; the new forest distance is composed of the old value and the cost for deletion or insertion, respectively. For map operations, however, special care must be taken to ensure that the vertical structure of the trees is obeyed, as required by the third condition of Equation 4.8. Please note that the horizontal tree structure, i.e. the left-to-right order, is automatically preserved by the dynamic programming procedure. Therefore, forests are decomposed into two parts:

- Part A: The sub-tree rooted at the current tree node i of $\mathbf{T}_1$ (or j of $\mathbf{T}_2$).
- Part B: All the other trees of the forest, located left of the current tree node.

In order to perform this decomposition, the index of the node at the boundary between the two parts needs to be known. Due to the postorder numbering scheme, the wanted node is the leftmost leaf node of the sub-tree rooted at the current node i. Let $\mathbf{T}[i]$ denote the sub-tree of $\mathbf{T}$ rooted at i. $\mathbf{T}[i]$ corresponds to Part A of the forest $\mathbf{T}[i'..i]$. Then, the index of the so-called leftmost leaf descendant of $\mathbf{T}[i]$ is denoted $l(i)$. For leaf nodes, $l(i)$ equals i. A sub-tree $\mathbf{T}[i]$ can likewise be written in forest notation as $\mathbf{T}[l(i)..i]$. Hence, Part B of the forest $\mathbf{T}[i'..i]$ (located to the left of i) is expressed as $\mathbf{T}[i'..l(i)-1]$ if $i' < l(i)$.

Since the goal is the computation of tree distances, the value of the left bound i' of the forest $\mathbf{T}[l(i_1)..i]$ is restricted to be a leftmost leaf descendant $l(i_1)$ of some tree node i_1. Hence, the right bound i of the forest $\mathbf{T}[i'..i]$ must be a value from the set of descendants of i_1, denoted $desc(i_1)$, i.e. part of the sub-tree rooted at $\mathbf{T}[i_1]$. Under the assumption that $i \in desc(i_1)$ and $j \in desc(j_1)$, the recursive forest distance computation can be formulated as in [SZ97]:

$$D_f^{min}(l(i_1)..i, l(j_1)..j) = \min \begin{cases} D_f^{min}(l(i_1)..i-1, l(j_1)..j) & + \quad c(i \mapsto \varepsilon) \\ D_f^{min}(l(i_1)..i, l(j_1)..j-1) & + \quad c(\varepsilon \mapsto j) \\ D_f^{min}(l(i_1)..l(i)-1, l(j_1)..l(j)-1) & \\ \quad + D_f^{min}(l(i)..i-1, l(j)..j-1) & + \quad c(i \mapsto j) \end{cases} \tag{4.10}$$

This equation can be viewed as the tree matching counterpart of the sequence matching formula (see Equation 4.5). The first two terms correspond to delete and insert operations, respectively. The third term corresponds to a map operation and consists of the distance between the forests left of i and j, of the distance between the sub-trees rooted at i and j and of the costs for the map operation $(i \mapsto j)$ itself. For the case that $l(i_1) = l(i)$ and $l(j_1) = l(j)$ the forests $\mathbf{T}_1[l(i_1)..i]$ and $\mathbf{T}_2[l(j_1)..j]$ are proper trees, so that there are no forests left of i and j. Thus, Equation 4.10 can be split into two cases:

(Case 1) If $l(i_1) = l(i)$ and $l(j_1) = l(j)$:

$$D_f^{min}(l(i_1)..i, l(j_1)..j) = D_t^{min}(i,j) = \min \begin{cases} D_f^{min}(l(i_1)..i-1, l(j_1)..j) & + \quad c(i \mapsto \varepsilon) \\ D_f^{min}(l(i_1)..i, l(j_1)..j-1) & + \quad c(\varepsilon \mapsto j) \\ D_f^{min}(l(i_1)..i-1, l(j_1)..j-1) & + \quad c(i \mapsto j) \end{cases} \tag{4.11}$$

(Case 2) Otherwise, i.e. $l(i_1) \neq l(i)$ or $l(j_1) \neq l(j)$:

$$D_f^{min}(l(i_1)..i, l(j_1)..j) = \min \begin{cases} D_f^{min}(l(i_1)..i-1, l(j_1)..j) & + \quad c(i \mapsto \varepsilon) \\ D_f^{min}(l(i_1)..i, l(j_1)..j-1) & + \quad c(\varepsilon \mapsto j) \\ D_f^{min}(l(i_1)..l(i)-1, l(j_1)..l(j)-1) & + \quad D_t^{min}(i,j) \end{cases} \tag{4.12}$$

where $D_t^{min}(\mathbf{T}_1[i], \mathbf{T}_2[j])$ or shorter $D_t^{min}(i,j)$ denotes the minimum tree distance between the sub-trees rooted at i and j, respectively.

The final goal is to determine $D_t^{min}(\mathbf{T}_1[|\mathbf{T}_1|], \mathbf{T}_2[|\mathbf{T}_2|]) = D_t^{min}(\mathbf{T}_1, \mathbf{T}_2)$. As we can see from Equation 4.12, this involves the computation of the tree distances $D_t^{min}(i,j)$ for all pairs of sub-trees $(\mathbf{T}_1[i], \mathbf{T}_2[j])$, $1 < i < |\mathbf{T}_1|$, $1 < j < |\mathbf{T}_2|$. However, some of these distances are already available from the computation of $D_t^{min}(i_1, j_1)$. These are the pairs of sub-trees whose roots i and j are in the paths

of $l(i_1)$ to i_1 and $l(j_1)$ to j_1, respectively. Thus, only the root node and all nodes with left siblings need separate computations. Formally, this set of nodes is called the *left-right keyroots* $LRKR(T)$ of $\mathbf{T}$, and is defined according to [SZ97] as:

$$LRKR(T) = \{k \mid \nexists\, k' > k \text{ such that } l(k') = l(k)\}$$

For the example of Figure 4.4, $LRKR(\mathbf{T}_1) = \{6, 8, 9, 13, 14, 15\}$ and $LRKR(\mathbf{T}_2) = \{4, 6, 8, 12, 13, 14\}$.

Please refer to [SZ97] or [TFLR03b] for a closer description of the implementation of Shasha and Zhang's algorithm, which computes a minimum tree edit distance using Equations 4.11 and 4.12. As [SZ97] shows, the complexity of this algorithm is:

$$O(|\mathbf{T}_1| \times |\mathbf{T}_2| \times min(depth(\mathbf{T}_1), leaves(\mathbf{T}_1)) \times min(depth(\mathbf{T}_2), leaves(\mathbf{T}_2)))$$

4.3.2 Semantic Tree Node Accuracy

After showing how mappings between pairs of ordered, labeled trees can be computed with an existing algorithm, this section presents an application of this technique to determine a novel, objective evaluation measure for speech interpretation. For this purpose, references in the shape of semantic tree annotations (see Section 4.1.2) are mapped to the semantic tree hypotheses of ODINS for a set of test utterances. Using the minimum tree edit distance mappings, we compute the degree of agreement between references and hypotheses in the same way as word or slot-value sequences (see Sections 4.2.2 and 4.2.3). Namely, an accuracy is determined by relating the erroneous map operations to the total number of units in the reference, only that now the units are tree nodes instead of words or slot-value pairs.

Consequently, we define a so-called semantic tree node accuracy Acc_n analogously to Equations 4.6 and 4.7 as

$$Acc_n = 1 - \frac{S_n + I_n + D_n}{N_n} \tag{4.13}$$

where C_n, S_n, I_n and D_n represent the counts of correct, substituted, inserted and deleted tree nodes, respectively, and $N_n = C_n + S_n + D_n$ denotes the total number of tree nodes of the reference trees. Since we make sure that the 'dummy' root nodes are always mapped to each other (see below), they are ignored during counting.

The direct use of the tree matching scheme of Section 4.3.1, and especially the use of the cost function of Equation 4.9, however, does not take the modeling structure of WTNH (see Chapter 2) fully into account. Specifically, tree nodes of different types, corresponding to different hierarchy levels, may be related to each other by a map operation, more precisely by a substitution. Yet, it is desirable to prevent such relations as they complicate separate evaluation of individual modeling levels (see below).

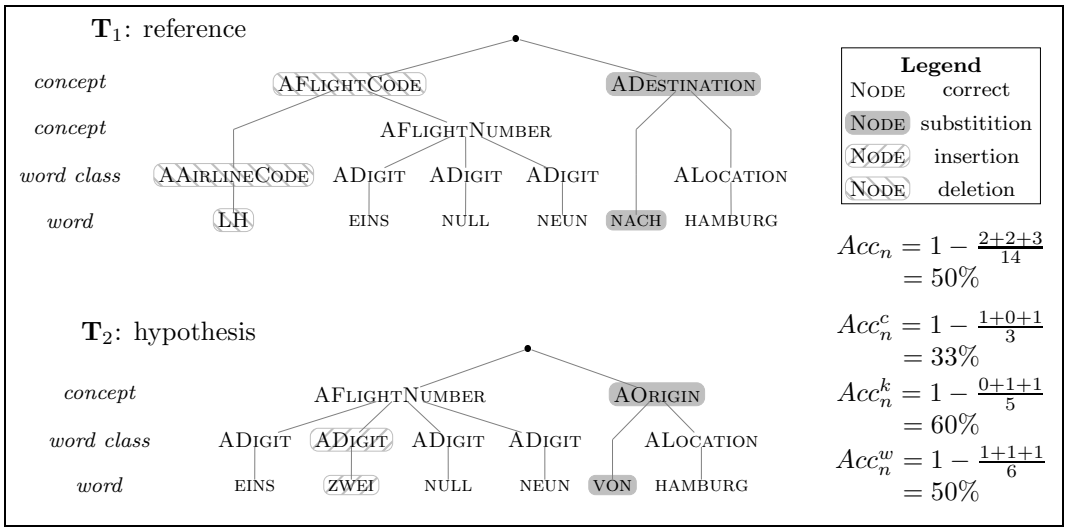


Figure 4.7: *Tree node accuracies of a decoded semantic tree hypothesis, matched with the annotated reference tree.*

Therefore, we extend the notion of tree matching from Section 4.3.1 so that only map operations between tree nodes of the same type are permitted. Algorithmically, this can be achieved by modifying the cost function of Equation 4.9 so that the type $\tau[i]$ of a tree node i is considered in addition to its symbol $t[i]$:

$$c(i \mapsto j) = \begin{cases} 0, & i,j \neq \varepsilon \text{ and } \tau_1[i] = \tau_2[j] \text{ and } t_1[i] = t_2[j] & \text{(correct)} \\ 4, & i,j \neq \varepsilon \text{ and } \tau_1[i] = \tau_2[j] \text{ and } t_1[i] \neq t_2[j] & \text{(substitution)} \\ \infty, & i,j \neq \varepsilon \text{ and } \tau_1[i] \neq \tau_2[j] & \text{(type substitution)} \\ 3, & i \neq \varepsilon \text{ and } j = \varepsilon & \text{(deletion)} \\ 3, & i = \varepsilon \text{ and } j \neq \varepsilon & \text{(insertion)} \end{cases} \tag{4.14}$$

This modification has the effect that a pair of insertion and deletion operations is always preferred over a type substitution, because it is less costly. The new cost function is also useful to ensure that the root nodes of the trees are always mapped to each other. Please note that Equation 4.14 implies that the computed tree distance is no longer a distance metric in the strict sense, as the triangle inequality (see Equation 4.2) does no longer hold.

Figure 4.7 illustrates the result of a match between a semantic tree reference and a semantic tree hypothesis. The trees are those shown in Figure 4.4 with additional node types, and were redrawn to reflect the tree levels. For this example, the counts are $S_n = 2$, $I_n = 2$, $D_n = 3$, and $N_n = 14$. Hence, the tree node accuracy yields $Acc_n = 1 - \frac{2+2+3}{14} = 50\%$. This accuracy value approximately corresponds to an intuitive rating of the interpretation goodness, since the flight number concept, three of its four digits and the location were recognized and interpreted correctly.

As mentioned above, the mappings between semantic trees can also be used to compute separate accuracy measures for individual tree node types τ, such as words, word classes or semantic concepts. By this, a more detailed error analysis can be performed. For this purpose, map operations of the minimum tree edit distance mapping are counted individually for each tree node type τ, yielding C_n^τ, S_n^τ, I_n^τ, D_n^τ and $N_n^\tau = C_n^\tau + S_n^\tau + D_n^\tau$. Hence, the tree node accuracy for nodes of type τ, denoted Acc_n^τ, is defined as:

$$Acc_n^\tau = 1 - \frac{S_n^\tau + I_n^\tau + D_n^\tau}{N_n^\tau} \tag{4.15}$$

The accuracy values for words ($\tau = w$), word classes ($\tau = k$) and concepts ($\tau = c$) for the example of Figure 4.7 are $Acc_n^w = 50\%$, $Acc_n^k = 60\%$ and $Acc_n^c = 33\%$, respectively.

Please note that the type-dependent accuracies Acc_n^τ are determined from the optimum match for the whole trees. This measure usually yields a lower value than one computed by an isolated, and hence vertically less restricted mapping between the considered tree levels. Specifically, $Acc_n^w \leq Acc_w$ for a given test set, i.e. the tree node accuracy for nodes of type word is at most as high as the accuracy of the word sequences contained in the semantic trees.

4.3.3 Experiment: Sequence vs. Tree Evaluation

In this section, an experiment is discussed which compares the tree matching based evaluation measure proposed in the previous section, and the standard sequence matching based measures defined in Section 4.2. A similar experiment, conducted with an earlier version of the airport information corpus, is described in [TFLR03b]. The evaluation measures are compared using different modeling setups for the considered one-stage speech interpretation task in the airport information domain. Specifically, the behavior at varying semantic model complexity is examined, because it can be expected that a tree matching based evaluation measure is more flexible than a sequence matching based one, and therefore especially suitable for models of higher complexity.

In order to clarify this expectation, lets consider an example. As illustrated in Figure 4.1, the slot-value pairs are in our case extracted in a post-processing step from the semantic tree produced by ODINS. Since the considered semantic trees generally consist of multiple levels of semantic category symbols, these categories need to be flattened to yield a semantic slot. For this purpose, we combine the ancestors of a leaf node in the tree into a single item. This yields the semantic slot, whereas the leaf node itself represents the value for this slot.

Figure 4.8 depicts the slot-value pair sequences for the semantic trees of Figure 4.7. Due to the deletion of the concept AFlightCode and the substitution of ADestination with AOrigin in the hypothesis, none of the slots match in this example. However, as discussed in Section 4.2.3, for a map operation to be treated

$\mathbf{S}_1$: reference

$\mathbf{S}_1^{slot}$	AFLIGHTCODE AAIRLINECODE	AFLIGHTCODE AFLIGHTNUMB ADIGIT	AFLIGHTCODE AFLIGHTNUMB ADIGIT	AFLIGHTCODE AFLIGHTNUMB ADIGIT	ADESTINATION	ADESTINATION ALOCATION
$\mathbf{S}_1^{val}$	LH	EINS	NULL	NEUN	NACH	HAMBURG

$\mathbf{S}_2$: hypothesis

$\mathbf{S}_2^{slot}$	AFLIGHTNUMB ADIGIT	AFLIGHTNUMB ADIGIT	AFLIGHTNUMB ADIGIT	AFLIGHTNUMB ADIGIT	AORIGIN	AORIGIN ALOCATION
$\mathbf{S}_2^{val}$	EINS	ZWEI	NULL	NEUN	VON	HAMBURG

$$Acc_{sv} = 1 - \frac{0+6+6}{6} = -100\%$$

Figure 4.8: *Slot-value pair sequences extracted from semantic trees of Figure 4.7.*

as correct or substitution, matching slots are required. Thus, the mapping of this example consists of insertions and deletions only, i.e. $I_{sv} = 6$ and $D_{sv} = 6$. With Equation 4.7 we therefore get $Acc_{sv} = 1 - \frac{0+6+6}{6} = -100\%$, i.e. the accuracy of the mapping even becomes negative. This result deviates largely from the accuracy of the equivalent tree match ($Acc_n = 50\%$) and from the intuitive rating of Figure 4.7.

The (admittedly extreme) example suggests that the two metrics can produce fundamentally different evaluation results. Although the sequence and tree based approaches both honor partially correct interpretations, the latter displays greater flexibility in finding correspondences between partially correct sub-structures, whereas the slot-value measure always requires fully matching structures in vertical direction.

For the experiment, different WTNH models incorporating varying degrees of semantic information were built. All WTNH contain identical acoustic-phonetic and lexical modeling levels as described in Section 5.6.2. The HLM are constructed as described in Chapter 5, using the training corpus of semantic tree annotations introduced in Section 4.1. Good-Turing and modified Kneser-Ney smoothing was employed to optimize HLM weights, as discussed in Section 5.7.

In order to reduce the degree of semantic information contained in the HLM, varying numbers of semantic concepts were deleted from the annotated tree corpus. The model building procedure was repeated after each deletion step, yielding 5 different HLM containing 41, 28, 19, 10 or 0 different semantic concept categories, respectively (the HLM with 0 concepts corresponds to a word class based language model). Hence, the phoneme, word and word class levels of all WTNH models are identical, whereas the concept and root levels differ. With decreasing numbers of semantic categories the complexity of the corresponding semantic model also decreases, specifically in vertical direction, and along with this the (average) vertical complexity of decoded semantic trees.

Figure 4.9 displays the accuracy values for all three evaluation measures (Acc_n,

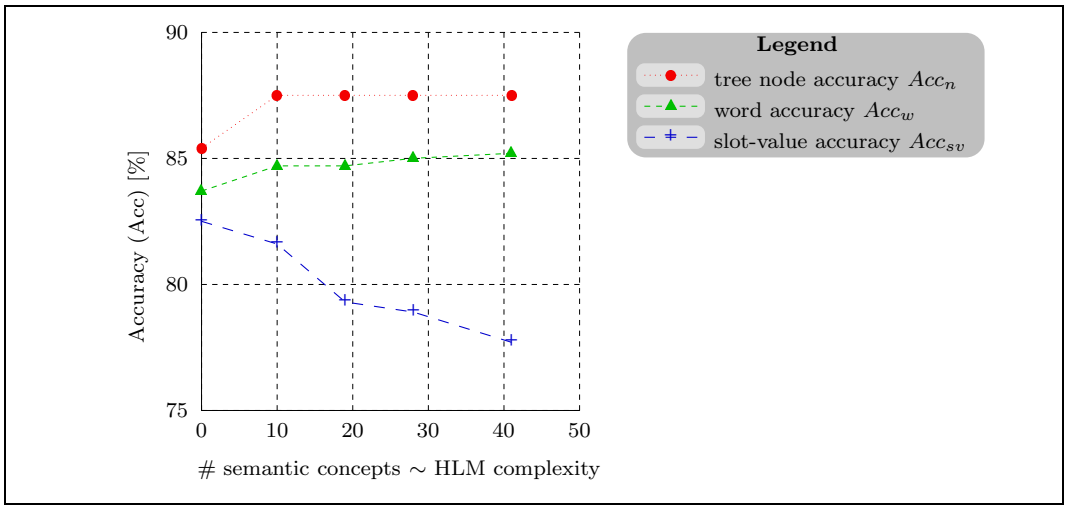


Figure 4.9: *Comparison of sequence and tree matching based evaluation measures for HLM of varying complexity.*

Acc_w and Acc_{sv}) in dependency of this HLM complexity, represented by the number of different semantic concepts contained in the HLM. The depicted accuracy values were computed on the evaluation set, after optimizing the free parameters (in this case the language model factor λ, see Section 5.9) on the cross-validation set. The parameter optimization was carried out separately for each evaluation measure and each WTNH model.

The experimental data confirm the expectation that the two semantic evaluation measures behave differently with increasing semantic tree complexity. The tree node accuracy tends to rise slightly from about 85% to 87.5%, whereas the slot-value accuracy decreases continually from 82.5% to about 78%. This is explained by the increasing vertical complexity of the semantic trees and the corresponding increase in *partially* correct vertical tree structures. As discussed in the example above, this partial correctness cannot be honoured by the sequence based measure, and hence Acc_{sv} shows decreasing accuracy.

Moreover, it can be noted that the word accuracy behaves more similar to the tree node accuracy, as it rises slightly from about 84% to about 85%. This conforms with the common expectation that the goodness of a speech interpretation system with respect to semantics is coupled to its word transcription capabilities.

Yet, the question which evaluation measure is more appropriate for a system such as ODINS may also depend on the task. If the following processing steps, e.g. the dialogue manager, are not capable to capitalize on partly correct semantic interpretations but rather depend on fully correct ones, slot-value accuracy may be rated as more suitable. For this thesis, however, it is assumed that succeeding modules can cope with such errors in the semantic tree structure. Therefore, tree

node accuracy is utilized as a primary evaluation measure throughout this work.

4.4 Language Model Perplexity

In the previous sections, several possibilities to evaluate ODINS's capabilities to process speech were discussed. Yet, the text processing part of a speech recognition or interpretation system, i.e. the language model, can also be assessed alone. The standard evaluation measure for the effectiveness of a language model is test-set perplexity [JMBB77]. In this section, a brief outline of the general notion of perplexity and a definition of computing test-set perplexity for HLM is given. For a more elaborate discussion on entropy and perplexity in language processing, see e.g. [JM00].

Perplexity is closely related to the information theoretic notion of entropy, suggested by Shannon [Sha48]. The entropy $H(p)$ of a signal or a random variable with probability distribution p is a measure of how much information is carried by the signal, in other words a measure of its randomness. Commonly, entropy is viewed as a lower bound on the number of bits required to encode an event in the optimal coding scheme.

If the actual probability distribution p of the signal is unknown, but a model $\hat{p}$ that approximates p is available, the cross-entropy $H(\hat{p})$ can be computed. The cross-entropy $H(\hat{p})$ is an approximation of the true entropy $H(p)$ and an upper bound on $H(p)$, whose goodness depends on how accurately $\hat{p}$ models p. For a discrete test signal $\mathbf{S}$ consisting of $|\mathbf{S}|$ symbols $s_1 \dots s_{|\mathbf{S}|}$, the cross-entropy is computed with $\hat{p}(\mathbf{S})$ as:

$$H(\hat{p}(\mathbf{S})) = -\frac{1}{|\mathbf{S}|} \log_2 \hat{p}(\mathbf{S})$$

Viewing each symbol s_i as an event and using the notion from above, $H(\hat{p}(\mathbf{S}))$ represents the average number of bits needed to encode each s_i.

The value 2^H is called perplexity. It can be interpreted as the average number of choices a random variable has to make. The counterpart of cross-entropy, i.e. the perplexity of a model $\hat{p}(\mathbf{S})$ measured over a test sequence $\mathbf{S}$, is often denoted as test-set perplexity ppl:

$$ppl(\mathbf{S}) = 2^{H(\hat{p}(\mathbf{S}))}$$

If $\hat{p}(\mathbf{S})$ is a word-based language model and $\mathbf{S}$ is a sequence of words representing a set of test sentences, $ppl(\mathbf{S})$ is an evaluation measure for that language model. It indicates how well the language model can predict the next word in an utterance, in terms of the average number of alternative words that need to be considered at the current position.

The lower the test-set perplexity of a language model, the better its ability to predict the next word. Hence, this measure can be used to compare language models built for the same task. Moreover, ppl can be used to compare the difficulty of

speech recognition tasks. Typical *ppl* values for n-gram language models on English or German texts range from about 10 to 1000, depending on the type of text. For the airport information application examined in this work, the n-gram *ppl* values are typically around 20.

The hierarchical language model considered in Chapter 5 represents a probability distribution $\hat{p}(\mathbf{T})$ over ordered trees $\mathbf{T}$ of semantic symbols. Similar to Section 3.1, $\mathbf{T}$ has L hierarchy levels, whereby each level $l = 1 \ldots L$ consists of a sequence $\mathbf{T}^l$ of $|\mathbf{T}^l|$ tree node symbols. However, the trees considered here only consist of words and higher modeling units, so that $\mathbf{T}^1$ denotes the word sequence represented by the leaf nodes of $\mathbf{T}$ (compare Section 5.1).

Letting $\mathbf{T}$ denote not a single tree but a whole test set as above, we define the cross-entropy $H(\hat{p}(\mathbf{T}))$ of the HLM $\hat{p}(\mathbf{T})$ and the corresponding test-set perplexity $ppl(\mathbf{T})$ as:

$$H(\hat{p}(\mathbf{T})) = -\frac{1}{|\mathbf{T}^1|} \log_2 \hat{p}(\mathbf{T})$$

$$ppl(\mathbf{T}) = 2^{H(\hat{p}(\mathbf{T}))}$$

Please note that we use the number of words $|\mathbf{T}^1|$ instead of the number of semantic symbols $|\mathbf{T}|$ to compute cross-entropy. Therefore, $ppl(\mathbf{T})$ relates to the average *word* probability assigned by the (hierarchical) language model, just like $ppl(\mathbf{S})$. This has the advantage that HLM with different semantic symbol sets can be compared, as will be done in Section 5.10.

Since HLM are constructed as weighted transition networks, $\hat{p}(\mathbf{T})$ is directly computed from the WTNH representation of an HLM $\hat{p}$. For this purpose, the (best) path of $\mathbf{T}$ through the WTNH is determined and all transition scores (log-likelihood values) along the path are summed up, which yields $\log \hat{p}(\mathbf{T})$. The transition networks visited along the path are defined by the tree node symbols contained in $\mathbf{T}$. Therefore, only local best path searches in single transition networks need to be carried out.

For perplexity computation a closed-vocabulary system is assumed. Therefore, unknown words in the test set are ignored, i.e. the $\hat{p}(\mathbf{T})$ and $|\mathbf{T}^1|$ are computed from known words only. As noted in [JM00], this means that the computed perplexity is no longer guaranteed to be greater than the true perplexity of the test set. Hence, the results must be interpreted carefully (see Section 5.8.1).

In a speech recognition or interpretation application, test-set perplexity is a glass box evaluation measure that doesn't consider the system as a whole like black box measures such as word accuracy or tree node accuracy, respectively. For speech recognition, it is generally assumed that language models with lower perplexity produce better speech recognition accuracy. Yet, numerous publications report that language models providing a large improvement in perplexity have yielded little or no improvement in word accuracy, especially when different types of language models

are considered (see [CBR98] for a more detailed discussion).

In this work, we utilize both test-set perplexity and tree node accuracy for evaluation. However, we generally rely on the black box measure, i.e. tree node accuracy, for parameter optimization (including HLM parameters). Although this requires considerably higher computational expenses, it ensures that the speech interpretation system is optimized on the whole.

Chapter 5

Hierarchical Language Models (HLM)

This chapter deals with the robust semantic modeling approach devised for the one-stage speech interpretation task pursued in this thesis. The so-called *hierarchical language model* (HLM) contains semantic and implicit syntactic knowledge represented by hierarchically structured, weighted transition networks. It corresponds to the upper part of the WTNH representation that was introduced in Chapter 2, and typically consists of a word, a word class and concept hierarchy levels (see Figure 2.2). The paths through the HLM network hierarchy structure define the possible semantic output trees of the one-stage decoder discussed in Chapter 3. HLM contain stochastic knowledge in the shape of weights on the transitions between network nodes. Using these weights, occurrence probabilities of semantic trees can be estimated. Together with the acoustic-phonetic and the lexical knowledge contained in the lower part of the network hierarchy, an HLM enables ODINS to compute the most likely semantic tree for a given speech signal. The main features of the hierarchical language modeling approach presented in this chapter were published in [TFLR05a].

In this work, it is assumed that the basic structure of a HLM is known, i.e. that its symbol vocabulary and the dominance relations between the symbols are defined manually. Approaches to automatically learn hierarchical language model structures from training data have been proposed (see e.g. [LY90, Car95, Che96]) but are not a focus of this thesis.

Yet, we still utilize data-driven language modeling techniques to build HLM. These methods generally assume the availability of appropriate training data. In our case, tree annotated training utterances are required. There are also methods to train hierarchical language models from word transcriptions and dominance relations only (see e.g. [HY03]). With such a technique, an explicit segmentation of semantic units can be omitted, which reduces annotation effort. Again, this is not a goal of this work. The HLM creation method pursued here relies on a training corpus fully

annotated with semantic trees. Such a corpus for an airport information application is described in Section 4.1.

In addition to data-driven techniques, we also employ rule-based approaches for certain parts of HLM. Rule-based language models of moderate size can be developed more rapidly and avoid the need for annotated training data. In contrast, data-driven techniques usually produce more robust language models and reduce the amount of required expert knowledge. In this work, we employ both rule-based and data-driven approaches to build suitable HLM. From this we generally expect to utilize the advantages and to possibly avoid the weaknesses of each method, and in particular to achieve good model performance with limited effort. Similar goals were recently pursued by [RH03, BJ04], for example.

In our notation, we discriminate between the terms *hierarchical language model* (HLM), *local language models* (LLM) and *language model* (LM). With the term HLM, we refer to an explicitly hierarchical description of a formal (tree) language consisting of a set of LLM. LLM are flat formal language descriptions consisting of terminal[1] and non-terminal symbols. Each non-terminal symbol refers to a LLM, so that the hierarchical structure of HLM is formed by these references. The LLM at the top of the hierarchy is called *root* LM.

With the term LM, we generally refer to a flat description of a formal language, such as a word language model for a speech recognizer. However, flat language modeling approaches are also applicable to LLM, if terminal and non-terminal symbols are (temporarily) treated equally (see Section 5.1). In this context, we mainly use the general term LM. To sum up, the terms LM and LLM both refer to flat, sequential models, whereas the term HLM refers to a hierarchical, tree-shaped model.

This chapter is organized as follows: In Section 5.1, a mathematical formulation of the hierarchical language modeling problem is presented. Section 5.2 then discusses properties of rule-based models and names suitable candidates for integration into the network hierarchy. The well-known n-gram LM and the so-called exact LM are described in Section 5.3 as representatives of the data-driven approach. For both types of LM, suitable smoothing techniques and their application within the uniform modeling framework are discussed. Due to the modularity of HLM, independent modeling decisions can be made for each LLM, as long as the selected modeling approach is representable within the WTNH framework. Section 5.4 presents a decision principle which facilitates the selection of LLM types.

When humans talk to spoken language processing systems in a natural way, a variety of phenomena can occur, each of which may have a negative influence on the system's performance. Section 5.5 describes how natural speech phenomena are modeled and handled by HLM. The basic modeling setup for the airport information

[1] The terminal symbols of HLM are no terminals when regarded in the context of the full network hierarchy, i.e. including the lexical and acoustic-phonetic hierarchy levels. In this case HMM states are the terminal symbols.

system examined in this thesis is explained in Section 5.6. The acoustic-phonetic and lexical model setups are described only briefly, as this work is not focussing on them. Section 5.7 presents experimental results on the influence of different HLM smoothing methods and the effects of varying smoothing parameters. In order to limit their complexity, LM usually only take a limited range of statistical dependencies into account. In Section 5.8, the range of HLM is compared to that of the standard word-based LM. Experimental results on HLM of varying dependency ranges are presented.

In speech recognition, it is common practice to adapt the likelihood distribution of the LM to the acoustic-phonetic likelihood values by a so-called LM factor. Section 5.9 describes how similar scaling can be applied to HLM and analyzes the resulting model experimentally. The results suggest that HLM themselves can be optimized by applying hierarchy level dependent scaling factors in order to balance the within-HLM weighting. As a further optimization parameter, the word insertion penalty is applied and investigated.

The ODINS system presented in this thesis can be regarded as a speech recognition system with additional semantic knowledge. This viewpoint raises the question if the use of this new knowledge source can increase recognition accuracy, even if this is not the original goal of this work. This question is pursued in Section 5.10.

5.1 Mathematical Formulation

A flat language model can be viewed as a likelihood distribution over symbol sequences. Given a sequence $\mathbf{S}$ of $|\mathbf{S}|$ symbols $s_1 \dots s_{|\mathbf{S}|}$ from an alphabet or vocabulary Σ, the likelihood $P(\mathbf{S})$ that this sequence occurs can be expressed as the product of the occurrence likelihood of the single symbols given their predecessors:

$$\begin{aligned} P(\mathbf{S}) &= P(s_1, \dots, s_{|\mathbf{S}|}) \\ &= P(s_1)P(s_2|s_1)P(s_3|s_1 s_2) \dots P(s_{|\mathbf{S}|}|s_1 \dots s_{|\mathbf{S}|-1}) \\ &= \prod_{i=1}^{|\mathbf{S}|} P(s_i|s_1 \dots s_{i-1}) \end{aligned} \tag{5.1}$$

Likewise, a HLM can be regarded as a likelihood distribution over ordered trees of semantic symbols. As described in Section 3.1, the hierarchical search problem requires computation of the likelihood $P(\mathbf{T})$ of ordered, labeled, constant-height trees $\mathbf{T}$ consisting of L hierarchy levels. Please note that in contrast to Chapter 3, where the complete WTNH is regarded, the current chapter deals with the 'higher' modeling levels. Therefore, we let L denote those levels that contain words and semantic objects, only.

The likelihood of a tree is expressed in terms of its tree levels $\mathbf{T}^l, l = 1 \dots L$ as:

$$P(\mathbf{T}) = P(\mathbf{T}^1, \mathbf{T}^2, \dots, \mathbf{T}^L)$$

Using the chain rule, this can be rewritten as:

$$P(\mathbf{T}) = P(\mathbf{T}^1|\mathbf{T}^2,\ldots,\mathbf{T}^L)P(\mathbf{T}^2|\mathbf{T}^3,\ldots,\mathbf{T}^L)\ldots P(\mathbf{T}^{L-1}|\mathbf{T}^L)P(\mathbf{T}^L)$$

As in Section 3.1, it is assumed that a tree level $\mathbf{T}^l$ only depends on the next higher tree level $\mathbf{T}^{l+1}$. Hence, $P(\mathbf{T})$ can be approximated by:

$$\begin{aligned} P(\mathbf{T}) &\approx P(\mathbf{T}^1|\mathbf{T}^2)P(\mathbf{T}^2|\mathbf{T}^3)\ldots P(\mathbf{T}^{L-1}|\mathbf{T}^L)P(\mathbf{T}^L) \\ &= P(\mathbf{T}^L)\prod_{l=1}^{L-1} P(\mathbf{T}^l|\mathbf{T}^{l+1}) \end{aligned} \tag{5.2}$$

As explained in Section 3.1, $\mathbf{T}^l$ describes a sequence of $|\mathbf{T}^l|$ tree nodes of level l. Each tree node has a symbol s_i^l from the vocabulary Σ. The symbols of a tree level can be segmented into $|\mathbf{T}^{l+1}|$ consecutive sub-sequences $\mathbf{S}_i^l$, so that Equation 5.2 becomes:

$$P(\mathbf{T}) \approx P(\mathbf{S}_1^L)\prod_{l=1}^{L-1}\prod_{i=1}^{|\mathbf{T}^{l+1}|} P(\mathbf{S}_i^l|s_i^{l+1}) \tag{5.3}$$

Each of the terms of Equation 5.3 are represented by the likelihood distribution of a LLM. The root LM describes the unconditional likelihood term $P(\mathbf{S}_1^L)$.

Using Equations 5.1 and 3.5, the terms corresponding to local language models are further decomposed into:

$$P(\mathbf{S}_1^L) = \prod_{j=1}^{|\mathbf{T}^L|} P(s_j^L|s_1^L\ldots s_{j-1}^L) \tag{5.4}$$

$$P(\mathbf{S}_i^l|s_i^{l+1}) = \prod_{j=A_i^l}^{B_i^l} P(s_j^l|s_{A_i^l}^l\ldots s_{j-1}^l, s_i^{l+1}) \tag{5.5}$$

Hence, Equation 5.3 becomes:

$$P(\mathbf{T}) \approx \prod_{j=1}^{|\mathbf{T}^L|} P(s_j^L|s_1^L\ldots s_{j-1}^L)\cdot\prod_{l=1}^{L-1}\prod_{i=1}^{|\mathbf{T}^{l+1}|}\prod_{j=A_i^l}^{B_i^l} P(s_j^l|s_{A_i^l}^l\ldots s_{j-1}^l, s_i^{l+1}) \tag{5.6}$$

For the one-stage decoding approach of this thesis, the likelihood distribution $P(\mathbf{T})$ of a HLM is represented by (the upper part of) a WTNH. A likelihood distribution $P(\mathbf{S}_i^l|s_i^{l+1})$ or $P(\mathbf{S}_1^L)$ of a single LLM is implemented by a single weighted transition network. Each of the likelihood terms $P(\ldots|\ldots)$ on the right hand side of Equation 5.6 corresponds to a weight on a transition between two network nodes.

Now we formulate the special case primarily examined in this thesis, where the semantic tree $\mathbf{T}$ consists of $L = 4$ hierarchy levels. These are denoted as word level

W, word class level **K**, concept level **C** and concept sub-level **C'**. Their symbols and sub-sequences are defined as follows:

$$\begin{array}{lllllll}
\mathbf{T}^1 & = & \mathbf{W} & = & w_1, w_2, \ldots, w_{|\mathbf{W}|} & = & \mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_{|\mathbf{K}|} \quad \text{(words)} \\
\mathbf{T}^2 & = & \mathbf{K} & = & k_1, k_2, \ldots, k_{|\mathbf{K}|} & = & \mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_{|\mathbf{C}'|} \quad \text{(word classes)} \\
\mathbf{T}^3 & = & \mathbf{C}' & = & c'_1, c'_2, \ldots, c'_{|\mathbf{C}'|} & = & \mathbf{C}'_1, \mathbf{C}'_2, \ldots, \mathbf{C}'_{|\mathbf{C}|} \quad \text{(sub-concepts)} \\
\mathbf{T}^4 & = & \mathbf{C} & = & c_1, c_2, \ldots, c_{|\mathbf{C}|} & = & \mathbf{C}_1 \quad \text{(concepts)}
\end{array}$$

For this case, the general formulation of the tree likelihood of Equation 5.3 becomes:

$$\begin{aligned}
P(\mathbf{T}) \approx\ & P(\mathbf{W}_1|k_1) \cdot P(\mathbf{W}_2|k_2) \cdot \ldots \cdot P(\mathbf{W}_{|\mathbf{K}|}|k_{|\mathbf{K}|}) \\
& \cdot P(\mathbf{K}_1|c'_1) \cdot P(\mathbf{K}_2|c'_2) \cdot \ldots \cdot P(\mathbf{K}_{|\mathbf{C}'|}|c'_{|\mathbf{C}'|}) \\
& \cdot P(\mathbf{C}'_1|c_1) \cdot P(\mathbf{C}'_2|c_2) \cdot \ldots \cdot P(\mathbf{C}'_{|\mathbf{C}|}|c_{|\mathbf{C}|}) \\
& \cdot P(\mathbf{C}_1)
\end{aligned}$$

With Equations 5.4 and 5.5 this is decomposed into:

$$\begin{aligned}
P(\mathbf{T}) \approx\ & \prod_{i=1}^{|\mathbf{K}|} \prod_{j=A_i^1}^{B_i^1} P(w_j|w_{A_i^1} \ldots w_{j-1}, k_i) \\
& \cdot \prod_{i=1}^{|\mathbf{C}'|} \prod_{j=A_i^2}^{B_i^2} P(k_j|k_{A_i^2} \ldots k_{j-1}, c'_i) \\
& \cdot \prod_{i=1}^{|\mathbf{C}|} \prod_{j=A_i^3}^{B_i^3} P(k_j|c'_{A_i^3} \ldots c'_{j-1}, c_i) \\
& \cdot \prod_{j=1}^{|\mathbf{C}|} P(c_j|c_1 \ldots c_{j-1})
\end{aligned}$$

As mentioned in the introduction of this chapter, a training corpus of semantic trees is the basis for building HLM. By extracting all occurrences of symbol sub-sequences for a unique semantic category, a flat training corpus is provided for the LLM corresponding to that category. The decomposition of a tree corpus into a set of sequence corpora enables the use of flat language modeling techniques, if terminal and non-terminal symbols are temporarily treated as the same type of symbol. Thereby, decisions about the (local) LM type can be made independently for each semantic category. However, a sequence corpus may also be omitted completely and the corresponding LLM be defined manually, if desired. This is typically the case if quality and/or quantity of training data is low. In the following three sections, we discuss rule-based and data-driven language modeling approaches for LLM, and criteria for their selection.

5.2 Rule-Based Language Modeling

We call a LM rule-based if its structure and, if applicable, its weighting is defined manually by a human expert. Such a definition is typically created by use of a formal grammar consisting of a set of grammar rules, or by a single grammar rule. Human experts mainly use their common knowledge and experience to write these rules, and no or little explicit, actual real-world data. Commonly used grammar formalisms are e.g. stochastic context-free grammars (see Section 2.2.3), weighted regular grammars and context-free rewrite rules (see Section 2.2.2).

Rule-based language modeling generally has the desirable property that it requires comparably little effort to achieve a relatively high coverage of the target language by defining a few general rules. However, a further increase of the initial coverage requires more and more effort from human experts, because generally an increasing number of rules are required to cover increasingly rare phenomena. Moreover, grammars of increasing size become more and more difficult to keep consistent, and the side effects of newly added rules become increasingly difficult to control.

Suitable grammar formalisms for HLM are basically all those representable within the WTNH framework. In terms of expressive power, these are all Chomsky type 2 and 3 grammars (see Section 2.2.1), i.e. context-free and regular grammars. For the airport information application examined in this work, rule-based LM are only used to model single semantic categories (compare Section 5.6.1). For this task, we found weighted context-free rewrite rules most suitable. If no sensible weighting can be given for a regular expression, a uniform weight distribution is assumed by default.

Network Representation

As discussed in Section 2.2.2, weighted context-free rewrite rules have equivalent representations as weighted finite-state automata, which in turn can be converted into weighted transition networks (for an example rule and a corresponding network, see Figure 1.1). This enables the integration of such rule-based language models into WTNH. For conversion of weighted context-free rewrite rules into weighted finite-state automata we employ the AT&T Lextools toolkit [Spr03, Spr99a, Spr99b]. This toolkit, which is based on the AT&T FSM Library (see Section 5.3.2), also performs automata minimization (see Section 2.2.2) during the conversion, in order to increase decoding efficiency.

5.3 Data-Driven Language Modeling

Data-driven language modeling approaches generally seem to require less manual work from human experts than rule-based models. Their basic principle is based on deriving LM automatically from annotated speech corpora. More specifically, the automatic training procedure determines the structure and weight distribution of the LM. Its structure is derived from the identities of the symbol sequences occurring

in the speech corpus, whereas the weighting of the model is computed from the corpus statistics in the shape of occurrence frequencies of symbol sequences. Another advantage of data-driven language modeling is that rarely occurring phenomena can be covered more easily than with rule-based techniques.

Yet, in order to achieve a broad coverage of the target language, substantial amounts of training data are necessary. Moreover, the training data needs to be representative for the target language. This implies that the frequency distribution of all phenomena matches the real-world distribution. As this can usually not be assumed, the frequency distribution is smoothed in order to increase LM accuracy. Hence, the effort for data-driven language modeling is not necessarily smaller than for rule-based language modeling. Yet, it needs to be considered that data collection and annotation can, at least partly, be carried out by non-experts.

In this thesis, two different data-driven approaches were pursued in order to create suitable LLM: The well-known *n-gram* model and the simple so-called *exact model*, which exactly covers the symbol sequences seen in the speech corpus. These two approaches are discussed in the following sections.

5.3.1 n-Gram Language Models

The most widely used language models in speech recognition are n-gram LM [Bak75, Jel76]. Their basic principle consists in assigning likelihood values to sequences of n symbols. Thereby, the dependency of the current symbol is limited to the $n-1$ previous symbols. The n-gram model likelihood $P_{ngram}(\mathbf{S})$ of a sequence $\mathbf{S}$ of $|\mathbf{S}|$ symbols $s_1 \dots s_{|\mathbf{S}|}$ from a vocabulary Σ can be expressed by

$$P_{ngram}(\mathbf{S}) = \prod_{i=1}^{|\mathbf{S}|} P(s_i|h_i^n) \tag{5.7}$$

where the term h_i^n denotes the history of the current symbol s_i. This history consists of the $n-1$ symbols preceding s_i:

$$h_i^n = s_{i-n+1} \dots s_{i-1}$$

Due to the limited dependency of n-gram LM, $P_{ngram}(\mathbf{S})$ is an approximation of the LM likelihood $P(\mathbf{S})$ of Equation 5.1.

The value of n is also denoted as order of the n-gram model. As a further consequence of its range limitation, an n-gram LM has the ability to cover symbol sequences of arbitrary length. This property is especially important for speech processing applications, because it enables the system to process spoken utterances of arbitrary length.

The number of n-gram parameters rises exponentially with n. In order to limit the computational load, and since the needed amount of training data depends on the number of parameters to estimate, the mostly used setting in practical speech recognition applications is $n = 2$ or $n = 3$. 1-grams are referred to as unigrams, 2-grams as bigrams and 3-grams as trigrams.

The n-gram parameters are trained from a corpus of symbol sequences. From the training corpus, the *count* $c(h_i^n s_i)$ of an n-gram $h_i^n s_i$ is determined, i.e. the number of times the n-gram occurs. These counts are the starting point for computing the n-gram likelihood distribution.

The basic n-gram LM is obtained when the likelihood $P(s_i|h_i^n)$ of an n-gram $h_i^n s_i$ is replaced with the maximum likelihood estimate $P_{ML}(s_i|h_i^n)$:

$$P(s_i|h_i^n) = P_{ML}(s_i|h_i^n)$$

This estimate is directly computed from the n-gram counts, after normalization with the counts of all n-grams with the same history h_i^n:

$$P_{ML}(s_i|h_i^n) = \frac{c(h_i^n s_i)}{\sum_{s_i} c(h_i^n s_i)} \tag{5.8}$$

Usually, this model is not used directly, because it doesn't perform well with insufficient amounts of training data.

To cope with this problem, the basic n-gram LM is smoothed. Smoothing comprises methods for adjusting the likelihood distributions of LM so that they perform more accurately. For this thesis, we split the smoothing problem into two tasks with respect to terminology, namely discounting and generalization.

Discounting is understood as the task of reducing the likelihood of the unreliable estimates from the observed counts, and of redistributing the freed probability mass. Generalization denotes the task of broadening the LM's coverage of the target language, by assigning non-zero probabilities to events, e.g. n-grams, which never occurred in the training data (unseen events). Exaggerated generalization causes generation of too many new events, which are increasingly confused with existing ones during decoding. The result of this phenomenon, which is denoted as overgeneration, is a degradation of LM performance. The tasks of discounting and generalization are often applied together, in the sense that the probability mass freed by discounting is redistributed among the unseen events created by generalization.

Smoothing is a central issue in statistical language modeling, and a large number of smoothing techniques for n-gram LM have been proposed. A comparative study of some of these can be found in [CG98]. For this thesis, several smoothing approaches were examined with regard to their suitability for HLM. In the following, two basic discounting techniques are outlined, namely additive discounting and Good-Turing discounting. Then, two smoothing schemes for n-gram LM are briefly reviewed, namely Katz smoothing and modified Kneser-Ney smoothing. At the end of this section, we discuss network representation of n-gram LM for their integration into HLM. In Section 5.3.2, we present methods for smoothing exact LM directly on the network level, again using additive discounting and Good-Turing discounting. Experimental comparisons of the applied smoothing techniques are then presented in Section 5.7.

The n-gram LM utilized in this thesis were all estimated with the SRI Language Modeling Toolkit (SRILM) [Sto02]. Furthermore, this toolkit was also employed to convert n-gram LM into weighted transition networks.

Discounting

If training data is sparse, it can be assumed that the distribution of events seen during training does not reflect the real distribution very well. Therefore, it is desirable to modify the frequency distribution of the training data in an appropriate way, and at the same time reserve some probability mass for unseen events.

A basic method of discounting is the so-called **additive discounting** [Lid20]. It is based on modifying the counts derived from training data by adding a constant value δ and renormalizing appropriately. This means that each n-gram is pretended to occur δ times more than it actually does. Hence, the maximum likelihood estimate from Equation 5.8 is replaced by the additive discounting estimate P_{add}:

$$P_{add}(s_i|h_i^n) = \frac{\delta + c(h_i^n s_i)}{\sum_{s_i} \delta + c(h_i^n s_i)} = \frac{\delta + c(h_i^n s_i)}{\delta\,|\Sigma| + \sum_{s_i} c(h_i^n s_i)} \tag{5.9}$$

The value of the additive discounting constant δ is typically around 1. In the case that $\delta = 0$, no discounting is performed. Thereby, the additive discounting estimate P_{add} degenerates into the maximum likelihood estimate P_{ML}. For large δ the likelihood distribution approaches a uniform distribution. The additive discounting method is simple, yet reported to perform comparatively poor [GC94].

A more refined and widely used technique is **Good-Turing discounting** [Goo53]. It is based on the assumption that the frequencies of observed events follow a binomial distribution. The Good-Turing method suggests that the observed counts c of events should be replaced by counts or frequencies c^*, where:

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{5.10}$$

In this equation, the term N_c denotes the frequency of all events in the distribution that were observed exactly c times, in other words the 'frequency of frequency'. The discount ratio d_c is defined as the ratio of the counts after and before discounting:

$$d_c = \frac{c^*}{c} = \frac{(c+1)}{c}\frac{N_{c+1}}{N_c} \tag{5.11}$$

For n-gram LM, the observed events are n-grams, so that $c = c(h_i^n s_i)$. The modified counts $c^*(h_i^n s_i)$ are converted to a likelihood by normalization with all events with the same history, as before. Hence, the Good-Turing estimate P_{GT} of an n-gram $h_i^n s_i$ which occurs $c(s_i|h_i^n)$ times is computed as:

$$P_{GT}(s_i|h_i^n) = \frac{c^*(h_i^n s_i)}{\sum_{s_i} c(h_i^n s_i)}$$

Using the discount ratio d_c, the Good-Turing estimate can also be expressed in terms of the maximum likelihood estimate:

$$P_{GT}(s_i|h_i^n) = d_c \cdot P_{ML}(s_i|h_i^n) \tag{5.12}$$

In practice, the observed values of N_c are not used directly, but replaced with smoothed values. As proposed by Katz [Kat87], it is assumed that counts larger than the so-called discount threshold c_{max} are reliable (in this work, the default value of the SRILM toolkit was used, i.e. $c_{max} = 7$). Hence, these counts are not discounted:

$$d_c = 1 \qquad \text{if} \quad c > c_{max}$$

For small counts $c \leq c_{max}$, the discount ratio $d_{c,katz}$ proposed by Katz is proportional to the Good-Turing discount d_c. Since the counts of unseen events should remain unchanged, d_c needs to be renormalized as follows:

$$d_{c,katz} = \frac{d_c - (c_{max}+1)\frac{N_{c_{max}+1}}{N_1}}{1 - (c_{max}+1)\frac{N_{c_{max}+1}}{N_1}}$$

Another practical issue is, that the frequency estimation of Equation 5.10 fails if $N_c = 0$. For this thesis, we adopted the solution of the SRILM toolkit to this problem: If $N_{c_{max}+1} = 0$, the discount threshold is decremented, i.e. $c_{max} = c_{max} - 1$. The decrementing is repeated until $N_{c_{max}+1} \neq 0$ or $c_{max} = 0$. Then, the discount ratios d_c for $c = 1 \ldots c_{max}$ are computed after Equation 5.11. If $N_c = 0$ during this computation, the corresponding discount ratio is set to $d_c = 1$. Similarly, values $d_c > 1$ are 'clipped' to $d_c = 1$. If no events occur less than or equal to $c_{max}+1$ times, the discount threshold becomes $c_{max} = 0$. In this case, Good-Turing discounting cannot be performed. As we want to avoid that unseen events (with a count of zero) are assigned a likelihood value of zero, we fall back to additive discounting in this case throughout this work.

Many smoothing techniques for n-gram LM are based on Good-Turing discounting (see [CG98] for an overview). In this thesis we employ the widely used Katz smoothing, which is briefly reviewed in the following section. However, we also utilize Good-Turing discounting directly to smooth the transition network representation of exact LM (see Section 5.3.2).

In the following section, we also outline a second n-gram LM smoothing technique, namely modified Kneser-Ney smoothing. This method is based on **absolute discounting** [NEK94], where a fixed discount $0 \leq D \leq 1$ is subtracted from each non-zero count. Hence, the discount ratio $d_{c,abs}$ for absolute discounting is:

$$d_{c,abs} = \frac{\max\{c - D, 0\}}{c}$$

As suggested in [NEK94], the discount D should be estimated from the frequencies N_1 and N_2 of n-grams with one and two counts, respectively, by:

$$D = \frac{N_1}{N_1 + 2N_2} \tag{5.13}$$

Originally, absolute discounting is used as a smoothing technique according to our terminology. As such, it also involves interpolation of higher- and lower-order n-grams. This aspect is outlined in the following section, where derived smoothing techniques are discussed.

Generalization

The basic n-gram model of Equation 5.8 is already able to generalize from the symbol sequences seen during training to new ones. This capability arises from its range limitation to $n-1$ previous symbols. However, because of data sparsity it is desirable that LM are able to assign non-zero likelihood values to a larger number of (n-gram) events, possibly even allowing arbitrary sequences of symbols from a given vocabulary Σ.

The generalization principle is usually based on combining n-gram LM with lower-order models. This combination can either be performed by linear interpolation, or by a recursive backoff method: If an n-gram has a count (and hence likelihood) of zero, the algorithm uses (backs off to) the likelihood of the $(n-1)$-gram instead. This likelihood is scaled with an appropriate backoff factor. If the $(n-1)$-gram also has a count of zero, the $(n-2)$-gram is used, and so on until the unigram s_i, which always has a non-zero count. Because of the final unigram backoff, this technique effectively produces generalized n-gram LM which cover arbitrary symbol sequences.

According to [KN95], the likelihood estimate of a **backoff n-gram** can generally be described with the following recursive equation:

$$P_{smooth}(s_i|h_i^n) = \begin{cases} P_{discount}(s_i|h_i^n) & \text{if} \quad c(h_i^n s_i) > 0 \\ \gamma(h_i^n) P_{smooth}(s_i|h_i^{n-1}) & \text{if} \quad c(h_i^n s_i) = 0 \end{cases} \tag{5.14}$$

This means that if an n-gram $h_i^n s_i$ occurs in the training data, its discounted likelihood estimate $P_{discount}(s_i|h_i^n)$ is used. For n-grams with zero count, the smoothed likelihood estimate of the backoff $(n-1)$-gram $P_{smooth}(s_i|h_i^{n-1})$ is used after scaling with a factor $\gamma(h_i^n)$. This factor corresponds to the backoff factor mentioned above. It is chosen so that the total number of counts in the distribution, and thereby its total probability mass, remains the same. Hence, the probability mass distributed among the lower-order models must correspond to the amount reserved during discounting.

An **interpolated n-gram** LM is generally represented by the equation:

$$P_{smooth}(s_i|h_i^n) = \lambda_{h_i^n} P_{ML}(s_i|h_i^n) + (1 - \lambda_{h_i^n}) P_{smooth}(s_i|h_i^{n-1})$$

In contrast to backoff LM, higher- and lower-order distributions are always combined by interpolation, not only when the higher-order count is missing. Yet, as shown in [CG98], higher-order counts may be less reliable, especially when they are low. Therefore, interpolated LM typically yield better performance than backoff

LM. According to [CG98], interpolated LM can also be expressed in the form of Equation 5.14. The backoff formulation is favored in this thesis, because it can easily be transfered into a transition network representation. This is discussed in the following section.

Katz smoothing [Kat87] directly yields a backoff n-gram after Equation 5.14. It utilizes the modified Good-Turing discounting described in the previous section. The discounted likelihood estimate is hence defined, analogously to Equation 5.12, as:

$$P_{discount}(s_i|h_i^n) = d_{c,katz} \cdot P_{ML}(s_i|h_i^n)$$

The scaling factor $\gamma(h_i^n)$ is derived from the probability mass to be distributed among all $(n-1)$-grams with history h_i^n. This value is computed by the difference between the total probability mass 1 and the accumulated discounted likelihood of all seen n-grams with history h_i^n. In order to obtain the fraction of this probability mass reserved for each individual $(n-1)$-gram, it is normalized with the total likelihood of all $(n-1)$-grams contained in the n-grams with history h_i^n:

$$\gamma(h_i^n) = \frac{1 - \sum_{s_i:c(h_i^n s_i)>0} P_{smooth}(s_i|h_i^n)}{1 - \sum_{s_i:c(h_i^n s_i)>0} P_{smooth}(s_i|h_i^{n-1})}$$

Modified Kneser-Ney smoothing [CG98] is based on Kneser-Ney smoothing [KN95], which in turn is based on the absolute discounting technique mentioned in the previous section. It yields an interpolated n-gram, which can be expressed with the backoff formulation of Equation 5.14. Instead of using a single discount parameter D as in absolute discounting, *modified* Kneser-Ney smoothing introduces three different parameters D_1, D_2 and D_{3+}, which are applied to n-grams with 1, 2, and 3 or more counts, respectively. Correspondingly, the discount ratio $d_{c,knmod}$ for an n-gram $h_i^n s_i$ is

$$d_{c,knmod} = \frac{c(h_i^n s_i) - D(c(h_i^n s_i))}{c(h_i^n s_i)}$$

where:

$$D(c(h_i^n s_i)) = \begin{cases} 0 & \text{if} \quad c(h_i^n s_i) = 0 \\ D_1 & \text{if} \quad c(h_i^n s_i) = 1 \\ D_2 & \text{if} \quad c(h_i^n s_i) = 2 \\ D_{3+} & \text{if} \quad c(h_i^n s_i) \geq 3 \end{cases}$$

The discounted likelihood estimate for the backoff Equation 5.14 is then defined as:

$$P_{discount}(s_i|h_i^n) = d_{c,knmod} P_{ML}(s_i|h_i^n) + \gamma(h_i^n) P_{smooth}(s_i|h_i^{n-1})$$

The discount parameters D_1, D_2 and D_{3+}, are estimated, analogous to Equation 5.13, by:

$$Y = \frac{N_1}{N_1 + 2N_2} \qquad D_1 = 1 - 2Y\frac{N_2}{N_1}$$

$$D_2 = 2 - 3Y\frac{N_3}{N_2} \qquad D_{3+} = 3 - 4Y\frac{N_4}{N_3}$$

The (modified) Kneser-Ney smoothing method does not assume that the likelihood values of lower-order n-grams are proportional to their absolute (maximum-likelihood) counts, as is the case e.g. with Katz smoothing. Instead, the likelihood of lower-order n-grams is adjusted corresponding to the number of different symbols that they follow.

[CG98] illustrates this notion with the following example: Consider a training corpus for a bigram LM that contains many occurrences of the word FRANCISCO, but always in conjunction with a single predecessor word SAN. Intuitively, the unigram backoff likelihood of FRANCISCO, i.e. the likelihood assigned to it when it occurs in different contexts, should be low. However, a backoff smoothing scheme that uses the absolute counts of FRANCISCO would assign a rather high unigram likelihood to it.

This can be avoided if the number of different predecessors is taken into account during generalization. For (modified) Kneser-Ney smoothing, this is done by defining the smoothed estimate P_{smooth} of the backoff $(n-1)$-gram of Equation 5.14 as

$$P_{smooth}(s_i|h_i^{n-1}) = \frac{N_{1+}(\bullet h_i^{n-1} s_i)}{N_{1+}(\bullet h_i^{n-1} \bullet)}$$

where $N_{1+}(\bullet h_i^{n-1} s_i)$ denotes the desired number of different symbols s_{i-n+1} (represented by the bullet) that precede the $(n-1)$-gram $h_i^{n-1}s_i$ at least once in the training data. This count is normalized with $N_{1+}(\bullet h_i^{n-1}\bullet)$, the sum of the counts $N_{1+}(\bullet h_i^{n-1} s_i)$ for all symbols s_i. Finally, the scaling factor $\gamma(h_i^n)$ of the backoff equation is defined as

$$\gamma(h_i^n) = \frac{D_1 N_1(h_i^n\bullet) + D_2 N_2(h_i^n\bullet) + D_{3+} N_{3+}(h_i^n\bullet)}{\sum_{s_i} c(h_i^n s_i)}$$

where $N_c(h_i^n\bullet)$ denotes the number of different symbols s_i with history h_i^n that occur c times in the training data. Thereby, 3+ is read as 'three or more times'.

Network Representation

After outlining different methods to smooth n-gram LM, we now show how these LM can be integrated into the hierarchical language modeling approach. For these purpose, they need to be represented as weighted transition networks. For backoff n-gram models, efficient network representations are known (see e.g. [RPB96, AMR03]). The backoff principle is implemented by so-called failure transitions, which realize a context change to an $(n-1)$-gram if the n-gram does not exist.

Figure 5.1 illustrates the basic transition network scheme of a bigram LM with three symbols s_1, s_2 and s_3. Each of these symbols is represented by a labeled node. The two triangular nodes symbolize the entry and exit nodes which are traversed at

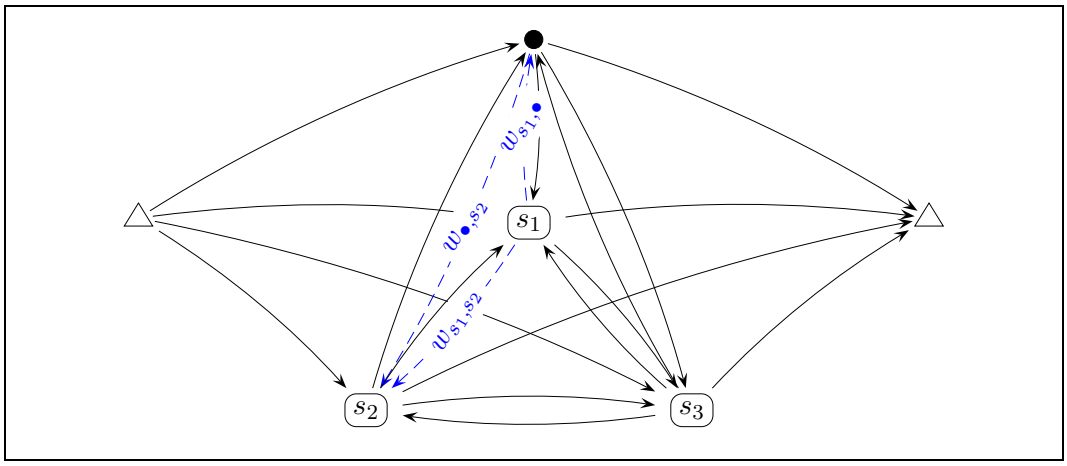


Figure 5.1: *Transition network representation of a bigram LM with vocabulary size* $|\Sigma| = 3$.

start and end of a symbol sequence (compare Figure 2.2). The null node ($\bullet$) is used as target for failure transitions.

A transition between two non-null nodes represents a bigram between the corresponding nodes, respectively their symbols[2]. Such a transition carries the log-likelihood value of the bigram as its weight. Context changes from bigrams to unigrams are implemented as failure transitions from non-null nodes to the null node. Their backoff likelihood is contained in the corresponding transition weight. Finally, unigrams and their likelihood values are realized as transitions from the null node to a non-null node. The network of Figure 5.1 contains all possible bigram transitions. In a practical network, only a fraction of all possible bigram transitions exist, namely those with non-zero likelihood.

If a bigram exists, there are two possible paths between the corresponding nodes, one direct path and one via the failure transition. The dotted transitions of Figure 5.1 show an example of this for the bigram $s_1 s_2$. The preference of the two possible paths is controlled via their scores, i.e. w_{s_1,s_2} for the direct path and $w_{s_1,\bullet} + w_{\bullet,s_2}$ for the backoff path. In the SRILM toolkit, which was also utilized for the conversion between n-grams and transition networks, this problem is solved by deleting the direct transition t_{ij} from node s_i to node s_j if it has a lower likelihood than the backoff path, i.e. if $w_{s_i,s_j} < w_{s_i,\bullet} + w_{\bullet,s_j}$.

5.3.2 Exact Language Models

In addition to n-grams, a second data-driven language modeling method was utilized in this work to represent LLM within HLM. A so-called exact LM exactly covers

[2]For this purpose, imagine that the entry and the exit node carry begin-of-sentence and end-of-sentence symbols, respectively.

the symbol sequences occurring in a training data set. Therefore, the likelihood $P_{exact}(\mathbf{S})$ of an exact LM for a symbol sequence $\mathbf{S}$ of length $|\mathbf{S}|$ is an exact description of Equation 5.1, and not an approximation like the n-gram likelihood $P_{ngram}(\mathbf{S})$ of Equation 5.7:

$$P_{exact}(\mathbf{S}) = \prod_{i=1}^{|\mathbf{S}|} P(s_i|s_1 \dots s_{i-1})$$

As for n-gram LM, the basic weight distribution is obtained by maximum likelihood estimation, based on the occurrence frequencies seen during training. As discussed later in this section, this distribution can be smoothed by discounting.

Because of their missing generalization capabilities, exact LM are clearly unsuitable as stand-alone, flat language models, unless huge amounts of training data are available. However, HLM inherently generalize to a certain extent because of their hierarchical structuring, as illustrated in this section. In order to prevent over-generation, exact LM were therefore used for certain parts of HLM, where generalization seems rather counterproductive.

Generalization

Due to their nature, exact LM themselves don't generalize at all. However, when considering the generalization abilities of the whole HLM, it has to be taken into account that the hierarchical structuring of semantic symbol sequences into semantic categories itself has a generalizing effect.

In order to illustrate this effect, let's assume for example that the utterances UM WIEVIEL UHR GEHT DAS FLUGZEUG (word-by-word translation: at what time goes the airplane) and WANN STARTET DIE MASCHINE (when starts the machine) occur in the training data, and that these utterances have both been annotated with the same concept sequence QTIME DEPARTURE PLANE, because they are semantically equivalent in the context of the considered application domain. If we now build exact LM for each of these three concepts, they contain two alternative symbol subsequences each. Using the notation of context-free rewrite rules, this is expressed as:

QTIME	→	UM WIEVIEL UHR	|	WANN
DEPARTURE	→	GEHT	|	STARTET
PLANE	→	DAS FLUGZEUG	|	DIE MASCHINE

Hence, as a result of the two utterances seen during training, the HLM is now able to produce 8 utterances. These correspond to the 2^3 possible combinations of word sequences contained in the concept sequence QTIME DEPARTURE PLANE, among them e.g. WANN GEHT DIE MASCHINE.

In general, the generalization effect caused by hierarchical structuring becomes the stronger the more semantic categories and the more hierarchy levels the HLM contains. Therefore, this effect must be taken into account when defining the hierarchy structure itself, but it also plays an important role for the decision which type of LM to use for a specific semantic category.

Network Representation

Transition network representations of exact LM can be obtained simply by creating an isolated network path for each symbol sequence of the training set. However, this type of representation is inefficient with respect to consumption of storage space, because it treats common partial paths separately. Therefore, many more nodes and edges are needed in comparison to an equivalent, compact version of the network. For the decoding process this means that more tokens are created and that these tokens are recombined later than necessary. Hence, compact network representations enable more space and time efficient decoding.

As outlined in Section 2.2.2, minimization of abstract automata can be performed automatically with the aid of methods from the finite-state automata theory. For this work, the AT&T FSM Library toolkit [MPR98, MPR03] was utilized to minimize FSA representations of exact LM. Due to the equivalence of Mealy and Moore machines (see Section 2.2.2), the minimized FSA can then be converted into equivalent transition networks.

In principle, these minimization and conversion operations could also be carried out on weighted representations of exact LM. Yet, in this work the weight distribution is transfered to the network *after* these operations. This procedure is used because we devised a general, data-driven method to convert unweighted transition networks into weighted ones. This technique, which is described in the following, is also applied to create stochastic versions of rule-based LM (see Section 5.4).

Discounting on the Network Level

In order to enable weighting and weight smoothing for a broader range of LLM, we perform these operations directly on the transition network representation of LLM. Thereby, all transition networks for which a suitable set of training data is available can be subjected to this method. In this section, exact LM are considered, but weighting and smoothing of rule-based LM is also performed (see Section 5.4).

As for n-gram LM, the weight estimation is based on counts. Here, counts directly apply to the network representation, more specifically to node transitions. Hence, we transfer the statistics of a training set to a network by walking the network path corresponding to each symbol sequence in the training set, and counting how often each transition is traversed. For this purpose, the transition network needs to be deterministic, as is the case for representations of exact LM.

Again, it must be assumed that the amount of training data is not sufficient to reflect the real distribution of events very well. Hence, smoothing is also desirable for exact LM. Due to the missing generalization capabilities of exact LM, the redistribution of probability mass among unseen events is not possible. Yet, we can still perform redistribution among seen events. In this case, smoothing is reduced to discounting.

In this work, we apply two different discounting techniques to exact LM, namely

additive discounting and Good-Turing discounting. Unlike for n-gram LM, where the basic events for discounting are n-grams, here the basic events are node transitions. Apart from this, discounting is performed as described in Section 5.3.1.

Let t_{ij} denote the transition from node with index i to node j, and $c(t_{ij})$ the count of a transition t_{ij}. Then, similar to Equation 5.9, the additive discounting estimate $P_{add}(t_{ij})$ of transition t_{ij} is computed by adding a constant value δ to the transition count, and normalizing over all transitions leaving node i:

$$P_{add}(t_{ij}) = \frac{\delta + c(t_{ij})}{\sum_j \delta + c(t_{ij})} \tag{5.15}$$

For Good-Turing discounting, the discount ratio $d_{c,katz}$ is computed as outlined in Section 5.3.1, using transition counts instead of n-gram counts. Generally, Good-Turing discounting can be applied to all transitions of a network at once, or separately for each node's outgoing transitions. The former method is used in this work, because only few nodes of practical networks have enough transitions to be able to apply the Good-Turing algorithm at all. From the modified transition counts $c^*(t_{ij}) = d_{c,katz}c(t_{ij})$, the Good-Turing estimate $P_{GT}(t_{ij})$ of transition t_{ij} is then computed, again normalizing over all transitions leaving node i:

$$P_{GT}(t_{ij}) = \frac{c^*(t_{ij})}{\sum_j c^*(t_{ij})}$$

Hence, the freed probability mass is redistributed uniformly.

5.4 Language Model Combination

As outlined in the introduction of this chapter, HLM are created by utilizing different rule-based and data-driven modeling techniques. By selecting the most suitable technique for each part of the HLM, we aim to improve model performance while reducing efforts. This goal is supported by the modularity of HLM and the underlying WTNH, which allows that independent modeling decisions are made for each network of the hierarchy. In this section, the selection of the best LM type for a given task, i.e. to describe a semantic category and its probability distribution, is discussed.

As explained in Sections 5.2 and 5.3, we utilized three types of LM as LLM in this thesis, namely weighted context-free rewrite rules as representative of rule-based language modeling, and n-gram LM and exact LM as data-driven models. Due to the lack of suitable metrics, an automatic procedure for selecting appropriate types for LLM couldn't be applied. However, some decision criteria and a corresponding decision principle were formulated. These criteria are based on the generalization abilities of the different LM types as discussed in the previous sections, on the complexity of the target language and on the amount of training data available for that target language. The decision principle for the three utilized types of LLM is shown

1. use **weighted context-free rewrite rules** if
   (a) the target language can be covered with a few simple rules, or
   (b) no or too little training data for a data-driven LM is available, or
   (c) essential events are not contained in the training data
2. else, use ***n*-gram LM** if
   (a) the symbol alphabet is large
   and the symbols occur in varying order, or
   (b) sequences of arbitrary length should be allowed
3. else, use **exact LM**

Figure 5.2: *Decision principle for the three utilized types of local language model.*

in Figure 5.2. The actual choices for the HLM used in the airport information system are then discussed in Section 5.6.1.

In addition to making hard decisions about the LLM type, it is also possible to combine rule-based and data-driven methods to estimate *one* LLM. Such an approach can for example be useful if the model structure can easily be given manually, but the corresponding weight distribution is not clear to the human expert. In this case, it may be desirable to transfer the statistics of an appropriate training corpus to the manually created, rule-based LM. Compared to a uniform weight distribution or one guessed by an expert, this more likely yields better model performance. In this work, the statistics transfer is performed with the method described in Section 5.3.2, except that the unweighted network corresponds to a context-free rewrite rule instead of an exact LM. As discussed in Section 5.3.2, the transfered weight distribution can also be smoothed.

The discussed method applies most suitably to Case 1(a) of Figure 5.2. In Case 1(b), there is no or probably not enough training data for weight estimation. In Case 1(c), the training data doesn't cover essential parts of the manually defined language by principle, so that data-driven weighting is rarely sensible.

5.5 Natural Speech Phenomena

We define natural speech as speech uttered by a human being in a natural way. This specifically excludes that textual information is predefined, as is the case with read speech. Moreover, speech is viewed as unnatural if speakers are limited with respect to vocabulary or speaking style.

Naturally spoken user utterances may contain a great variety of phenomena.

processing level	phenomena
acoustic	pauses, breathing, laughter, coughing, throat-clearing, lip-smacking
lexical	filler words (hesitations, mumbling), cut-off words
semantic	repetitions, self-corrections, ungrammatical utterances

Table 5.1: *Examples of natural speech phenomena.*

These pose a possible threat to the performance of a speech interpretation system if they aren't modeled appropriately. Some of the phenomena occurring in natural speech are listed in Table 5.1, categorized by the processing level on which they are typically treated (first). In the following, we will explain how some of these effects are modeled in this work.

When humans speak naturally, they sometimes utter sounds which can be viewed as speech sounds but not as proper words. We call these phenomena filler words. Typical examples are sounds that are the result of periods in which the user is unsure what to say next, also called hesitations. These speech sounds may also have a semantic relevance, e.g. the German sound '*mhm*', which can mean an acknowledgement of something the communication partner (be it a human or a machine) said. In this thesis, filler words are modeled on the lexical level (see Section 5.6.2) by assigning several variants as different pronunciations to a lexical entry denoted <*fill*>. Because even humans can be unsure of their meaning, filler words are treated as semantically irrelevant.

Longer pauses and non-speech effects like breathing sounds, laughter and coughing are considered on the acoustic-phonetic level by the specially trained HMM *nib* and *p:* (see also Section 5.6.2). Special lexical units were created for both of these HMM, denoted <*nonsp*> and <*silence*>.

Another typical phenomenon of natural speech are incompletely spoken words, so-called cut-off words. This phenomenon is difficult to cope with, since cut-off words are likely to be confused with proper in-vocabulary words, because they actually consist of proper speech sounds until they are cut off. One possible approach to this problem is to cover cut-off words with a generic lexical model. In Chapter 6 of this thesis, such a modeling technique is presented and examined. Even if its main purpose is the coverage of out-of-vocabulary (OOV) words, its generic lexical nature makes it suitable for the cut-off word problem.

In order to take such natural speech phenomena into account, they also require integration into HLM. Due to training data sparsity, we combine filler, non-speech and silence 'words' in a single filler word class in this work, so that they ultimately appear to the language model as one unit. As mentioned above, cut-off words are treated as OOV words. Apart from this, the modeled natural speech effects are treated like any other symbol in the HLM, i.e. they are generally contained in several LLM on different hierarchy levels. Please note that the filler word class is

regarded as syntactically and semantically irrelevant. Therefore, its occurrences are ignored during evaluation.

Some natural speech effects only occur on the syntactic-semantic modeling level. These are e.g. self-corrections or ungrammatical, i.e. syntactically incorrect, utterances. The HLM approach considered in this work has a certain robustness against such phenomena by design, specifically due to:

- robust semantic modeling without explicit syntactic analysis
- generalization and discounting of observed events, using hierarchical structuring and local language modeling
- combination of data-driven and rule-based language modeling techniques
- OOV modeling, which enables a fragmentary analysis, similar to key-word spotting, where only semantically relevant utterance parts are modeled

5.6 Test System Setup

This section describes the system setup for the experimental results presented in this chapter. The experiments were all carried out with a test system for an airport information application. The collection and annotation of speech data for this domain was outlined in Section 4.1.

The most important properties of this speech corpus are repeated briefly here: The collected speech data comprises around 2700 utterances from 32 subjects, with 5.5 words per utterance on average. Recordings were carried out with a close-talking microphone in two different environments, a quiet laboratory and a driving car. The corpus was split into three parts (see Section 4.1.2): A training set with the utterances of 20 subjects, an evaluation set and a cross-validation set with 6 subjects each. The OOV rates are 2.2% on the evaluation set and 1.5% on the cross-validation set.

5.6.1 Hierarchical Language Model Setup

Unlike the acoustic-phonetic model (see following section), HLM are solely estimated from the airport information corpus. The semantic tree annotations of its training set comprise 4 hierarchy levels (see Section 4.1.2): One word and word class level each, and two semantic concept levels. The word, word class and concept levels consist of 594, 12 and 41 unique symbols, respectively.

Several different basic HLM setups can be generated by masking certain hierarchy levels in the annotations. The default speech interpretation system setup contains all of the 4 hierarchy levels. In order to build a speech recognition system for comparison with the speech interpretation system (see Section 5.10), the concept levels are hidden.

The different types of LLM are selected by means of the decision principle presented in Section 5.4 as follows: Case 1(a) of Figure 5.2 applies to the semantic concepts for flight number and flight code. Using the regular expression operators defined in Section 2.2.2, the corresponding context-free rewrite rules are written as:

$$\begin{array}{lcl} \text{AFlightNumber} & \rightarrow & \text{ADigit ADigit ADigit ADigit?} \\ \text{AFlightCode} & \rightarrow & \text{AAirlineCode AFlightNumber} \end{array}$$

Case 1(c) occurs for virtually all word classes, because only part of all digits, times, airports, airline codes etc. that are relevant for the airport information domain are contained in the training data. The manually added rules contain around 30 new words for the lexicon.

Due to the variability of the root symbol sequence, n-gram LM (Case 2) are utilized as root model of both word class and concept based HLM by default. In Section 5.8, where varying n-gram orders of the root LM are examined, we also evaluate HLM with exact LM at the root level. All other than the mentioned semantic categories are implemented as exact LM (Case 3).

Word alternatives within word classes are weighted uniformly, in order to simulate real-world conditions. The likelihood distribution of all other exact networks is derived from corpus statistics (with *exact network* we denote a network generated from an exact LM or from a context-free rewrite rule). Exact networks and n-gram LM are smoothed with one of two different methods each. For the former, we apply additive discounting or Good-Turing discounting as discussed in Section 5.3.2. n-Gram LM are estimated with the SRILM toolkit and subjected to Katz smoothing or modified Kneser-Ney smoothing as outlined in Section 5.3.1.

5.6.2 Lexical and Acoustic-Phonetic Model Setup

Although acoustic-phonetic and lexical models are not a special focus of this work, their quality needs to reflect the state-of-the-art of speech interpretation systems. Thereby, the transferability of the results of this thesis can be judged adequately.

In order to model word pronunciation, a linear lexicon is created by manual pronunciation transcription. Due to the reasons discussed in Section 2.3.4, only few pronunciation variants are added for frequently occurring deviations from the canonical form. The final lexicon contains about 620 words, 30 of which don't occur in the training corpus, but are added manually during word class member completion (see previous section).

As speaker-independent, triphone HMM are today mostly used for similar tasks, these are selected as acoustic-phonetic models. Due to the explicitly hierarchical structure of WTNH, the context dependency of triphone models cannot extend word boundaries, as was explained in Section 2.3.4. Therefore, intra-word triphones are employed instead of the better-performing cross-word triphones. Still, intra-word triphones offer a good compromise between training effort and consideration of context effects.

A basic requirement for the acoustic-phonetic models of the test system is speaker-independence. However, the acoustic speech data collected for the airport information domain is, according to experience, too small to train reasonably well performing speaker-independent models. A common approach to circumvent this problem, which was also utilized here, consists of first estimating base models on a larger set of speech data from one or several other data collections, and then using the in-domain data to adapt these models to the target domain. The danger of this method is a potential mismatch between the in-domain data and the base data, which might degrade system accuracy. There are numerous factors for a possible mismatch, e.g. microphone characteristic, recording environment (background noise), but also speaking styles (read speech, spontaneous speech) or accents, whose phenomena can also have a negative effect on the acoustic model performance.

For this work, the base models are trained on a speech data collection from the German Verbmobil project [Wah00]. The used training set consists of about 11000 spontaneously spoken utterances from about 600 different speakers, recorded in an appointment negotiation scenario. The acoustic conditions of these recordings match those of a quiet laboratory environment. Therefore, only the laboratory environment part of the airport information corpus (compare Section 4.1.1) is used to adapt the base models to the target domain, but no speech data from in-car recordings. The adaptation is performed by a combined Maximum A-Posteriori (MAP) and Maximum Likelihood Linear Regression (MLLR) training, carried out with the HTK toolkit [YEK+02]. The probability densities of the phoneme HMM are modeled with Gaussian mixtures with diagonal covariance matrices. The HMM training yields about 25000 Gaussian mixture components in total.

The set of 50 German phonemes is derived from the definitions of the Verbmobil project. In addition to these phonemes, the HMM set contains a silence model (*p:*), a short-pause model (*sp*) and a model for non-speech effects (*nib*, see also Section 5.5). The HMM are three- or four-state left-right models, except for the short-pause model. This consists of a single state, tied to the middle state of the silence HMM. An additional skip transition renders the short pause model optional. Therefore, it can simply be appended to all lexical entries.

The preprocessing stage of ODINS is similar as in other state-of-the-art speech recognition systems. It computes 12 MFCC components, which together with energy, delta and acceleration coefficients yield 39-dimensional feature vectors.

Because an investigation of runtime performance optimization is not a goal of this thesis, all experiments presented in this thesis were carried out with a conservative setting of the beam pruning threshold (compare Section 3.2.2) of $t = 800$.

5.7 Smoothing Experiments

In order to analyze the influence of smoothing techniques and parameters on the performance of our speech interpretation system, a number of different concept based HLM were generated.

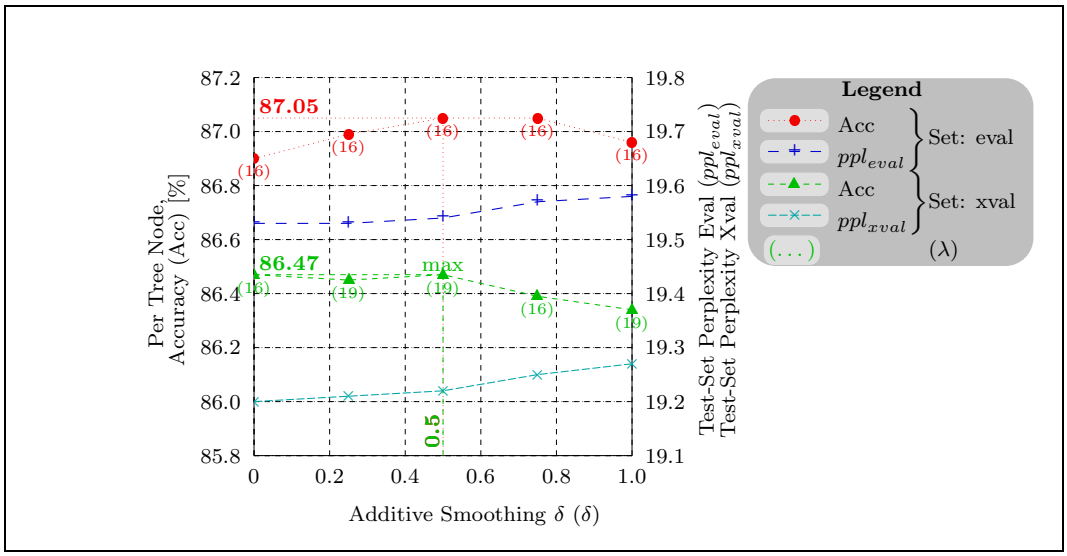


Figure 5.3: *Influence of additive discounting parameter δ on tree node accuracy and HLM test-set perplexity.*

In the first experiment, the influence of the additive discounting constant δ for exact networks (see Equation 5.15) is examined. As discussed in Chapter 4, the tree node accuracy Acc_n is used as primary evaluation measure. Additionally, we report results for the test-set perplexity $ppl = ppl(\mathbf{T})$ of HLM. The experiment is carried out with a Katz smoothed bigram LLM at the root level.

Figure 5.3 illustrates the results of Acc_n and ppl on the evaluation (eval) and cross-validation (xval) sets for settings of $\delta = 0 \ldots 1$. Tree node accuracy and test-set perplexity are shown on the left and right y axes, respectively. In brackets below the data points, those settings of the LM factor λ (see Section 5.9.1) are shown, that yield optimum accuracy. On the cross-validation set, the best accuracy of 86.47% is obtained with $\lambda = 19$ at $\delta = 0.5$. Surprisingly, the same accuracy can be achieved with $\lambda = 19$ at $\delta = 0$. At this setting of δ, discounting is effectively disabled. Because this is generally not desirable, $\delta = 0.5$ is selected for the standard system configuration. The curve of Acc_n on the evaluation set confirms this choice. There, the maximum accuracy of 87.05% is also obtained at $\delta = 0.5$. Compared to the case where no smoothing is performed ($\delta = 0$), this corresponds to an absolute accuracy gain of 0.15% or to a relative error rate reduction of $Err_{rel} = -1.2\%$. The optimum LM factor is $\lambda = 16$ in both cases.

The test-set perplexities correlate fairly well with the tree node accuracies, although the best values at $\delta = 0$ are marginally better than the perplexities at $\delta = 0.5$, both for evaluation and cross-validation. Generally, the absolute differences in perplexity are rather small.

In the second experiment we investigate the influence of the different smoothing techniques (ST). Again, bigram LLM are employed at the root level. As explained in

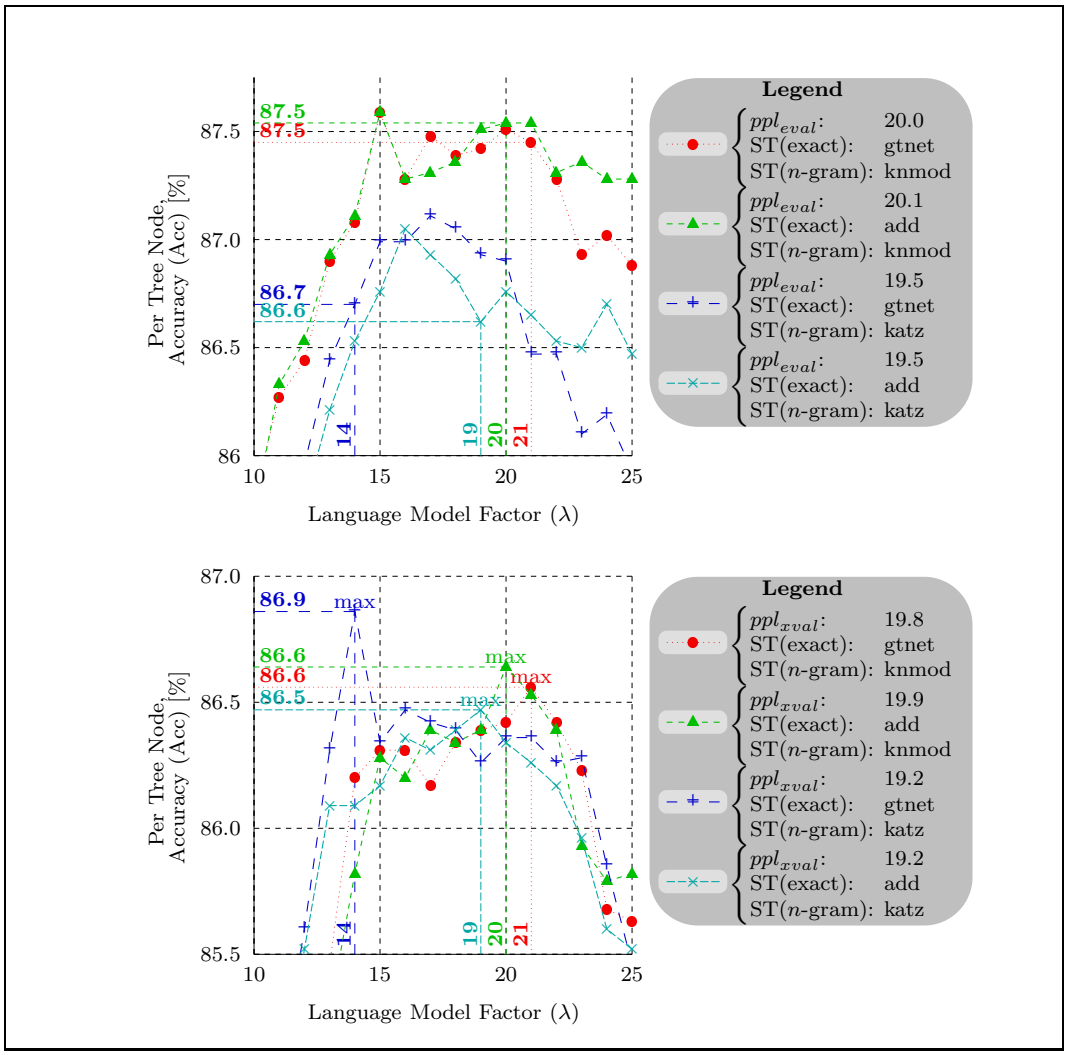


Figure 5.4: *Combinations of exact network discounting and n-gram LM smoothing, evaluation (top) and cross-validation (bottom).*

Section 5.6.1, exact networks are smoothed with additive discounting (add) or Good-Turing discounting (gtnet). For n-gram LM, Katz smoothing (katz) or modified Kneser-Ney smoothing (knmod) is applied.

Figure 5.4 depicts separate plots for evaluation and cross-validation sets with four curves each, corresponding to the possible combinations of each of the two smoothing techniques for exact networks and n-gram LM. Both plots show the total tree node accuracy Acc_n for different LM factor settings between 10 and 25. On the cross-validation set, the absolute maximum of $Acc_n = 86.9\%$ is obtained at $\lambda = 14$ for the HLM combining Katz n-gram smoothing with Good-Turing discounting of exact networks. However, examining the accuracy values around $\lambda = 14$ reveals

that this maximum is rather unstable. Better compromises between λ-stability and optimum accuracy are instead obtained by combining modified Kneser-Ney n-gram smoothing with additive or Good-Turing exact network discounting. Both of these HLM achieve $Acc_n = 86.6\%$ at their optimum LM factor settings.

This choice is confirmed on the evaluation set, where both knmod-smoothed HLM outperform the katz-smoothed HLM significantly over the whole range of λ. For the optimum λ settings from the cross-validation set, both knmod-smoothed HLM achieve accuracy values of $Acc_n = 87.5\%$. In contrast, the katz-smoothed HLM only obtain $Acc_n = 86.6\%$ and 86.7%. This corresponds to relative error rate reductions of $Err_{rel} = -6.7\%$ and -6.0%, respectively. The accuracy differences between additive and Good-Turing exact network discounting on the evaluation set are negligible. Therefore, we selected the theoretically more attractive Good-Turing discounting for our standard system configuration. Generally, it should be noted that the accuracy curves of Figure 5.4 suggest a significant mismatch between the characteristics of evaluation set and cross-validation set.

The test-set perplexity value for each HLM is listed in the legends of the two plots. Surprisingly, this evaluation measure indicates an advantage of Katz smoothing over modified Kneser-Ney smoothing on both sets (19.2 against 19.8/19.9 on the xval set, 19.5 against 20.0/20.1 on the eval set). This also contradicts the study of Chen and Goodman [CG98], where modified Kneser-Ney smoothing is reported as the clear winner of the examined smoothing techniques, based on perplexity evaluation. However, the scope and scale of their experiments differs substantially from the ones presented here.

5.8 Range of Language Model Dependencies

An important characteristic of a language model is how much of the previously uttered it takes into account to determine what is currently being said. In other words, how many previous symbols a LM considers in order to decide which symbol is currently the most likely one. This section deals with dependency ranges of flat and hierarchical LM. First, we illustrate and discuss how statistical dependencies are modeled with both approaches. Then, the influence of the root level LM range on HLM performance is examined in Section 5.8.1.

The range of the basic n-gram LM is limited to $n-1$ previous symbols. As has been mentioned in Section 5.3, values of $n>3$ are seldom used for speech interpretation tasks, because of the exponential rise of the number of n-gram parameters. Therefore, statistical relations between a larger number of symbols than captured by the standard n-gram LM are called long-range dependencies.

A hierarchically structured LM enables modeling of such long-range, statistical dependencies by hierarchical combination of several shorter dependencies. This characteristic of HLM is clarified in Figure 5.5, where the utterance IN WELCHEM BEREICH KOMMT BA NEUN NULL SIEBEN SECHS AN (word-by-word translation: in which area comes BA nine zero seven six at) along with its semantic tree is de-

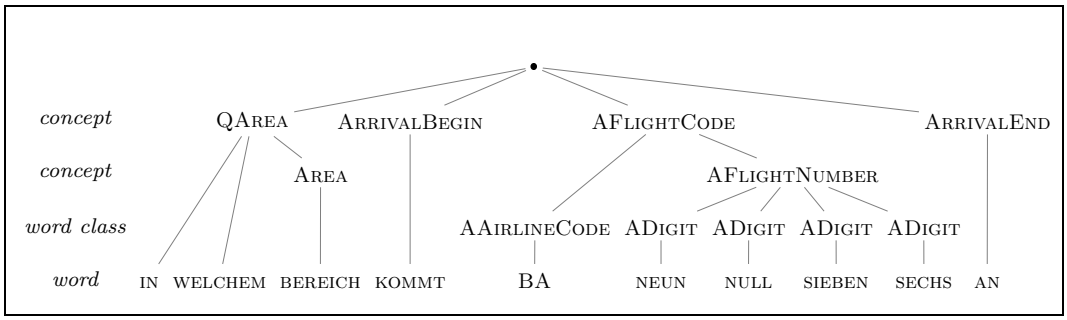


Figure 5.5: *Example for a long-range statistical dependency, which requires a 7-gram LM on the word or word class level, but only a trigram on the concept level.*

picted. In this utterance, there is a syntactic-semantic relation between the words KOMMT and AN, which together mean 'arrive'. On the word level, a 7-gram would be needed to model this statistical relation with a flat n-gram LM. However, only the 3 symbols ARRIVALBEGIN AFLIGHTCODE ARRIVALEND are necessary to cover the same statistical dependency on the root level, since 5 of the words to be spanned are contained in a single semantic category AFLIGHTCODE.

Besides the root level of HLM, such statistical relations can also occur within the hierarchical structure, e.g. between the digits in the concept AFLIGHTNUMBER of Figure 5.5. If generalization within the semantic category is not important, the statistical relations can be modeled directly by use of an exact network, whose dependencies span the whole modeled symbol sequence.

As outlined in Section 5.6.1, the HLM examined this thesis employ n-gram LM at the root level only. At this level, many different symbol sequence variations are observed, so that the n-gram model's ability to cover symbol sequences of arbitrary constitution and length is especially useful. Within HLM, the variations are much smaller. Therefore, and due to the reasons discussed in Sections 5.4 and 5.6.1, exact networks are used there. As these networks are capable of modeling long-range statistical dependencies, the dependency range of the root LM is mainly responsible for the range of the whole HLM.

It should be considered, however, that a hierarchical structuring of LM can be a disadvantage if direct relations between words are important. By subsuming several words or other semantic objects in a semantic category, the statistical relation between semantic objects in different categories is no longer modeled directly. Instead, the lower-level dependency is replaced by a higher-level dependency, which only directly affects the higher-level semantic categories. Because of this, HLM cannot yield optimum performance if their structure obliterates lower-level statistical dependencies which significantly contribute to the speech interpretation process. Consequently, the potential of an HLM fundamentally depends on the definition of its semantic categories and their hierarchical structuring.

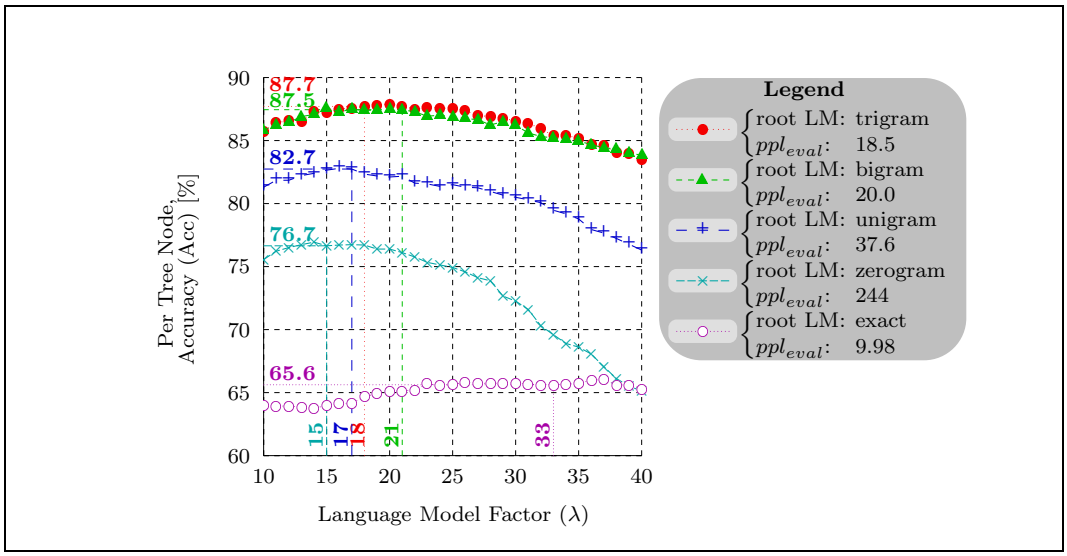


Figure 5.6: *Total tree node accuracy for varying n-gram orders of the root language model on the evaluation set.*

5.8.1 Experiment

In this section, the effect of different dependency ranges on the performance of HLM and the whole speech interpretation system is examined. For this purpose, the n-gram order of the root LM of a concept based HLM is varied between 0 (uniform symbol likelihood) and 3 (dependencies on up to 2 previous symbols). Furthermore, an HLM with exact root LM is also included in this experiment. As explained in Section 5.7, Good-Turing discounting of exact networks and modified Kneser-Ney smoothing of n-gram LM is utilized as standard configuration for this and other experiments.

Figure 5.6 gives an overview of the resulting tree node accuracy for the different model setups on the evaluation set when varying the LM factor λ (see Section 5.9.1) between 10 and 40. Each numerically highlighted value of Acc_n corresponds to the optimum λ setting of an HLM configuration obtained on the cross-validation set (see Figure A.1 in Appendix A). In the legend of the plot, the test-set perplexities of the examined HLM are listed.

First of all, it can be noted that, as expected, an increase of the n-gram order from $n = 0$ to $n = 2$ substantially increases the accuracy of the speech interpretation system from 76.7% to 87.5%. Accordingly, the test-set perplexities of HLM improve with increasing dependency range, from 244 for the zerogram to 20.0 for the bigram. Although the perplexity value of 10 promises the opposite, the use of an exact root LM dramatically deteriorates accuracy. This accuracy degradation is a consequence of the model's inability to cover unseen events. The obverse perplexity result can be explained with the used perplexity computation scheme, where unknown parts

of the utterance are simply ignored (see Section 4.4). Therefore, this result cannot be interpreted reasonably in this case.

The increase of the root LM order from bigram to trigram causes a significant gain in perplexity, which improves from 20.0 to 18.5. However, the corresponding tree node accuracy gain is rather small (0.2% absolute). An explanation for this could be that long-range dependencies play a secondary role in the tested system. As another possibility, a root model range of 3 might be still too small to take specific long-range dependencies sufficiently into account. Furthermore, training data sparsity might be the cause for the lacking performance gain of the trigram model. Because of the small accuracy difference between bigram and trigram root LM, bigrams are selected for the standard system configuration.

5.9 Likelihood Balancing

The language model factor (LM factor) λ is an essential parameter in practical speech recognition, because it balances the likelihood values of acoustic-phonetic model (AM) and LM heuristically against each other. This balancing is performed by a linear scaling of the likelihood distribution of the LM. In this section, the use of similar scaling techniques for HLM are studied. For this purpose, the basic approach for word-based LM, as used in speech recognition systems, is reviewed in Section 5.9.1. Section 5.9.2 deals with a metric for closer examination of the effects of different scaling parameters, the so-called insertion-deletion ratio. The application of the standard scaling parameter to HLM is outlined in Section 5.9.3. Experimental results, which suggest an extension of the basic scaling technique, are presented. In Section 5.9.4, we define the novel method and measure its effects on system performance. The results indicate a further possibility of improvement by use of an additional offset parameter. Its influence is examined in Section 5.9.5.

5.9.1 Language Model Factor

The necessity for an adjustment of likelihood values arises because in contrast to LM, the typically employed, HMM-based AM produce no real probabilities. Instead, HMM emission probabilities are represented by values on the probability density functions of mixture densities. Yet, both AM and LM likelihood values contribute to the total likelihood of a spoken utterance. Due to their different value ranges they cannot be combined directly.

The LM factor λ tries to eliminate this mismatch by linearly transforming the log-likelihood values of LM into the range of likelihood values of AM[3]. In addition to adapting the ranges of likelihood values, λ simultaneously balances the relative influence of AM and LM on the decoding process. The higher the value of λ, the more the recognition results are dominated by the likelihood distribution of the

[3]The scaling could likewise be applied to the AM likelihood values, transforming them into the LM's value range.

LM. This dominance has the effect that likely word transitions with unlikely acoustic matches are preferred over less likely word transitions with more likely acoustic matches. This property can be utilized to compensate quality differences between AM and LM heuristically, by giving more influence to the model of higher quality. This avoids recognition errors caused by the lower-quality model.

The following outlines how λ is formally applied in a speech recognition system: Let $\mathbf{X}$ denote a sequence of $|\mathbf{X}|$ acoustic feature vectors $x_1 \dots x_{|\mathbf{X}|}$ and $\mathbf{W}$ denote a sequence of $|\mathbf{W}|$ words $w_1 \dots w_{|\mathbf{W}|}$. Similar to Equation 3.2 of Chapter 3, the problem of finding the optimum word sequence $\mathbf{W}^*$ is solved with the maximum a-posteriori equation:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \left[P(\mathbf{X}|\mathbf{W})P(\mathbf{W})\right] \tag{5.16}$$

The term $P(\mathbf{X}|\mathbf{W})$ is estimated by the acoustic-phonetic and lexical models, $P(\mathbf{W})$ is represented by the LM. In practice, the likelihood values are not used directly, because the computation of the multiplicative likelihood components would lead to numerical underflow. Instead, their logarithm (log-likelihood, also called score) is used. Hence, Equation 5.16 becomes:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} \left[\log(P(\mathbf{X}|\mathbf{W})P(\mathbf{W}))\right]$$

The LM factor λ is applied as an exponent of the stochastic LM $P(\mathbf{W})$, so that the search problem becomes:

$$\begin{aligned} \mathbf{W}^* &= \arg\max_{\mathbf{W}} \left[\log(P(\mathbf{X}|\mathbf{W})P(\mathbf{W})^{\lambda})\right] \\ &= \arg\max_{\mathbf{W}} \left[\log(P(\mathbf{X}|\mathbf{W})) + \lambda \log(P(\mathbf{W}))\right] \end{aligned} \tag{5.17}$$

Using Equation 5.1, the λ-scaled LM log-likelihood of Equation 5.17 is decomposed into:

$$\begin{aligned} \lambda \log(P(\mathbf{W})) &= \lambda \log(P(w_1)) + \lambda \log(P(w_2|w_1)) \\ &\quad + \ldots + \lambda \log(P(w_{|\mathbf{W}|}|w_1 \dots w_{|\mathbf{W}|-1})) \\ &= \sum_{i=1}^{|\mathbf{W}|} \lambda \log(P(w_i|w_1 \dots w_{i-1})) \end{aligned} \tag{5.18}$$

Assuming that $\mathbf{X}$ is segmented into $|\mathbf{W}|$ consecutive sequences of feature vectors $\mathbf{x}_1 \dots \mathbf{x}_{|\mathbf{W}|}$, the AM log-likelihood of Equation 5.17 is expressed as:

$$\log(P(\mathbf{X}|\mathbf{W})) = \sum_{i=1}^{|\mathbf{W}|} \log(P(\mathbf{x}_i|w_i)) \tag{5.19}$$

By inserting Equations 5.18 and 5.19 into Equation 5.17, the search problem is finally expressed as:

$$\begin{aligned}\mathbf{W}^* &= \arg\max_{\mathbf{W}} \Big[\log(P(\mathbf{X}|\mathbf{W})P(\mathbf{W})^{\lambda})\Big] \\ &= \arg\max_{\mathbf{W}} \left[\sum_{i=1}^{|\mathbf{W}|} \log(P(\mathbf{x}_i|w_i)) + \lambda \log(P(w_i|w_1 \ldots w_{i-1}))\right] \end{aligned} \tag{5.20}$$

From this equation it can be seen that LM scaling is performed by multiplying each word log-likelihood with λ. This means that when we represent LM as transition networks, their scaling is performed by λ-multiplication of the log-likelihood values on all transitions.

The optimum value of λ is determined experimentally, because it depends on the type and quality of the used AM and LM. Typical values for a speech recognizer using triphone HMM and n-gram LM are in the range of $5 \ldots 15$.

As a side-effect of balancing the likelihood distributions of AM and LM, λ also affects the average length of recognized words. The reason for this phenomenon can be explained as follows: For large values of λ, the LM likelihood values become comparatively small, so that a traversal from one word to another one is more 'costly' than staying within a word. In this case, longer words are favored over shorter ones on average. Therefore, word deletion errors occur more often than word insertions. On the other hand, small values of λ favor the likelihood values of the AM, so that word traversals occur more often.

5.9.2 Insertion-Deletion Ratio

In this section, we investigate the phenomenon described at the end of the previous section, namely that λ influences the average length of a recognized word. For this purpose, we introduce a metric that describes the effects of likelihood scaling on word length.

As for the word accuracy measure outlined in Section 4.2.2, we assume that evaluation is carried out by matching the decoder hypotheses for a set of test utterances against their corresponding reference transcriptions. As a result of such an evaluation, the total number of correct, substituted, inserted and deleted words in the test set is known. These numbers are denoted by C_w, S_w, I_w and D_w, respectively. As mentioned in Section 4.2.2, the total number of words in the reference transcriptions N_w^{ref} can be expressed by:

$$N_w^{ref} = C_w + S_w + D_w \tag{5.21}$$

Together with the duration of the test utterances, N_w^{ref} is a measure for the average word length in the reference transcriptions. Likewise, the total number of word hypotheses N_w^{hyp} is a measure for the average length of the hypothesized words. It can be written as:

$$N_w^{hyp} = C_w + S_w + I_w \tag{5.22}$$

For our purpose, we are not interested in the absolute word length but in the relation between reference and hypothesis. This could be described by the ratio between N_w^{hyp} and N_w^{ref}. Since the numbers of correct and substituted words occur in both of these terms, they can be omitted. Hence, our measure is defined as the ratio of the number of word insertions and the number of word deletions in the optimum match between the reference transcriptions and recognizer hypotheses of a test set. The measure is denoted as *word insertion-deletion ratio* IDR_w:

$$IDR_w = \frac{I_w}{D_w} \tag{5.23}$$

The appearance of recognition errors depends on the properties and interactions of different modeling levels. Therefore, a simple relationship between the different kinds of errors cannot be given. However, we can expected that the decoder should recognize about as many words as were spoken to achieve its maximum accuracy, i.e. when:

$$N_w^{hyp} \approx N_w^{ref}$$

By first applying Equations 5.21 and 5.22, and then Equation 5.23, this becomes:

$$D_w \approx I_w \qquad\qquad IDR_w \approx 1$$

Thus, the value of IDR_w should be around one at the value of λ that yields the maximum word accuracy Acc_w. For small values of λ the cost of word traversals is low, so that more insertions than deletions occur on average and the value of IDR_w becomes larger than one. Large λ values render word traversals more costly, so that relatively many deletions occur and IDR_w becomes smaller than one. For increasing values of λ, the IDR_w curve should decrease monotonously.

5.9.3 Application of Language Model Factor to HLM

The basic problem of balancing AM and LM likelihood also exists for the speech interpretation task examined in this work. This section outlines how the basic likelihood scaling scheme of Section 5.9.1 is similarly applied to HLM. Furthermore, we show how the notion of the insertion-deletion ratio measure introduced in the previous section is extended to HLM, and discuss experimental results.

The formulation of the speech recognition problem given in Equation 5.16 can be generalized to the hierarchical search problem of speech interpretation, as shown in Section 3.1. In this section, we specifically consider the standard HLM of this work, which consists of three main hierarchy levels and one sub-level. As in Section 5.1, the main levels represent words, word classes and semantic concepts, which are denoted **W**,**K** and **C**, respectively. The symbols on the concept sub-level are denoted **C'**. For this HLM structure, the problem of finding the most likely output tree $\mathbf{T}^*$ from the feature vectors **X** is written, according to Equation 3.6, as:

$$\mathbf{T}^* = \underset{\mathbf{W},\mathbf{K},\mathbf{C}',\mathbf{C}}{\arg\max} \left[P(\mathbf{X}|\mathbf{W})P(\mathbf{W}|\mathbf{K})P(\mathbf{K}|\mathbf{C}')P(\mathbf{C}'|\mathbf{C})P(\mathbf{C})\right] \tag{5.24}$$

Similar as flat LM, HLM are scaled by applying λ as an exponent to their likelihood distribution. Hence, Equation 5.24 becomes:

$$\mathbf{T}^* = \underset{\mathbf{W},\mathbf{K},\mathbf{C}',\mathbf{C}}{\arg\max} \left[P(\mathbf{X}|\mathbf{W})[P(\mathbf{W}|\mathbf{K})P(\mathbf{K}|\mathbf{C}')P(\mathbf{C}'|\mathbf{C})P(\mathbf{C})]^{\lambda} \right] \tag{5.25}$$

By use of Equations 5.6 and 5.19 it can be shown that, similar to Equation 5.20, the scaling applies to all transitions within the language model. In the case of WTNH representations of HLM this means that all transition scores within networks that correspond to a semantic category, including the root network, are multiplied by λ.

Similar to Section 5.9.2, the effects of the LM factor on HLM shall be described. The metric is again based on results of evaluation test utterances, but this time the tree matching based evaluation technique of Section 4.3 is utilized. If C_n, S_n, I_n and D_n represent the counts of correct, substituted, inserted and deleted tree nodes, the number of nodes in the reference tree N_n^{ref} and the number of nodes in the hypothesized tree N_n^{hyp} are expressed by:

$$N_n^{ref} = C_n + S_n + D_n \qquad N_n^{hyp} = C_n + S_n + I_n$$

Similar to the word related measure of Equation 5.23, we define the *tree node insertion-deletion ratio* IDR_n as:

$$IDR_n = \frac{I_n}{D_n}$$

Again it is expected that N_n^{ref} approximately equals N_n^{hyp} at the λ value that yields optimum accuracy (this time tree node accuracy Acc_n), and hence $D_n \approx I_n$ and $IDR_n \approx 1$.

Furthermore, we can extend this expectation to individual hierarchy levels: As discussed in Section 4.3.2, the numbers of correct and erroneous symbols counted separately for each tree node type $\tau \in \{w, k, c\}$ are denoted C_n^τ, S_n^τ, I_n^τ and D_n^τ. Consequently, hierarchy level dependent versions IDR_n^τ of the insertion-deletion ratio measure are defined for words, word classes and concepts as:

$$IDR_n^w = \frac{I_n^w}{D_n^w} \qquad IDR_n^k = \frac{I_n^k}{D_n^k} \qquad IDR_n^c = \frac{I_n^c}{D_n^c} \tag{5.26}$$

If the numbers of nodes of each type are about equal in reference and hypothesis trees, we get:

$$IDR_n^w \approx 1 \qquad IDR_n^k \approx 1 \qquad IDR_n^c \approx 1$$

However, it should be noted that this expectation is in fact only justified if an HLM's likelihood distribution is balanced in itself. Generally, optimizing λ with regard to maximum tree node accuracy only ought to equalize the number of tree nodes of all types together, but not the nodes of individual types.

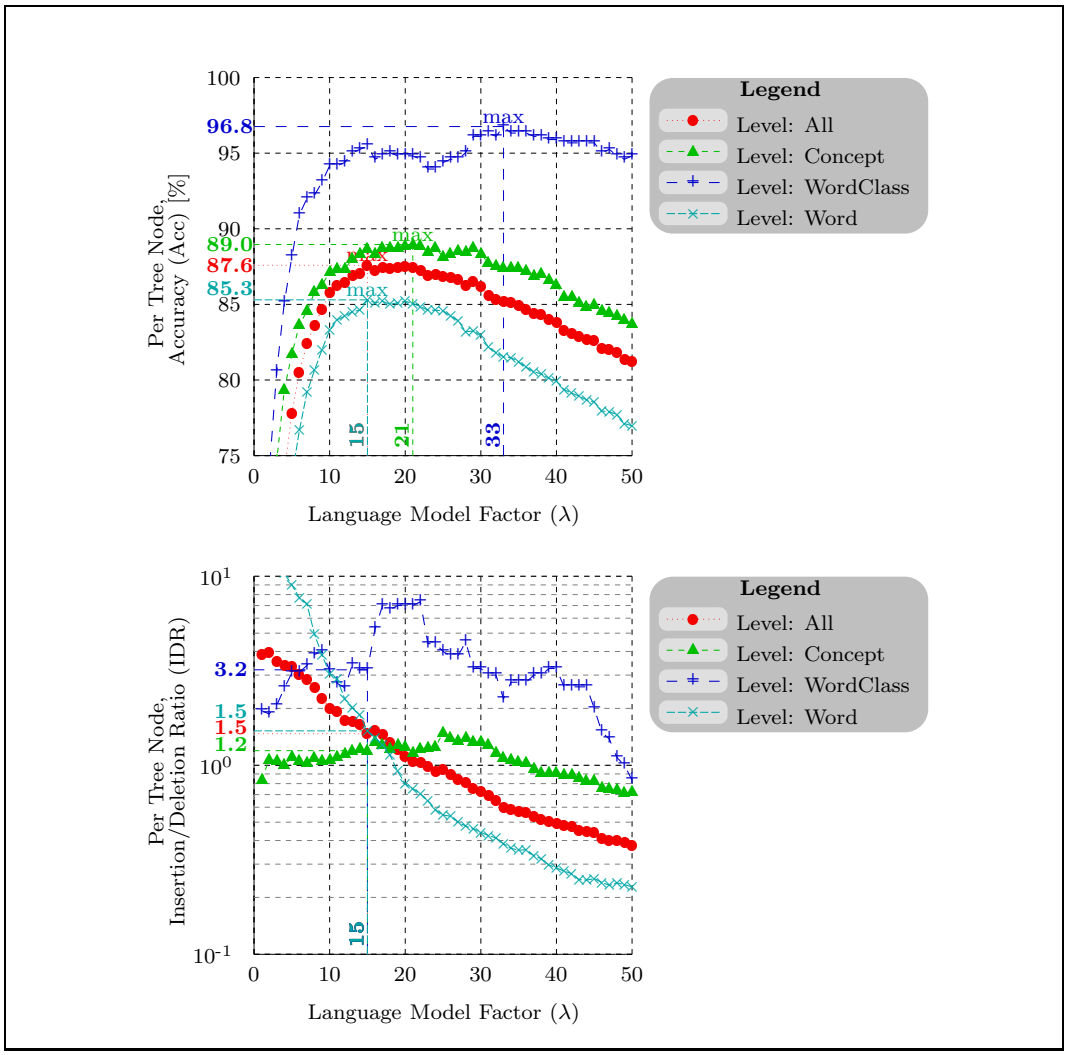


Figure 5.7: *Top: Tree node accuracy for all and individual node types for varying λ on the evaluation set. Bottom: Insertion-deletion ratio for the identical setup.*

Experiment

The influence of λ on the performance of our one-stage speech interpretation system on the evaluation set of the airport information application is depicted in Figure 5.7. The experiment was conducted with the standard system configuration elaborated in previous sections, i.e. a 4-level HLM with bigram root LM, modified Kneser-Ney smoothing and Good-Turing discounting.

In the upper plot of Figure 5.7, the total tree node accuracy Acc_n for all hierarchy levels as well as the individual tree node accuracies Acc_n^τ (compare Equations 4.13 and 4.15) for nodes of type word ($\tau = w$), word class ($\tau = k$) and concept ($\tau = c$)

are shown for $\lambda = 0 \ldots 50$. Additionally, the value of the absolute maximum of each curve is depicted. Firstly, it can be noted that accuracy levels differ consistently among all hierarchy levels. While the maximum value at the word level of 85.3% is well below the total accuracy of 87.6%, the concept accuracy is significantly higher with 89.0%. The word class level even achieves 96.8%. These differences are the result of the varying vocabulary sizes on the different hierarchy levels. With only about 10 unique word classes, the confusability between these units is much lower than the confusability between about 600 unique words. Clearly, structural dependencies between levels also play an important role, so that this can only be a general rule.

Another observation from the upper plot of Figure 5.7 is that the λ settings which yield maximum accuracy vary substantially between individual node types (15 for words, 21 for concepts and 33 for word classes). In order to analyze this phenomenon, the tree node insertion-deletion ratios for all and individual node types, IDR_n and IDR_n^τ (compare Equation 5.26) are depicted in the lower plot of Figure 5.7 for the same range of λ. The IDR curves are plotted on a logarithmic scale, which is well suited to illustrate differences in the considered ratio. The plot also shows the IDR values for $\lambda = 15$, i.e. the point of maximum Acc_n.

As discussed earlier in this section, we would expect $IDR_n \approx 1$ at this point, and furthermore $IDR_n^\tau \approx 1$. Whereas the former applies fairly well ($IDR_n = 1.5$), especially the word class value ($IDR_n^w = 3.2$) deviates substantially from our expectation. On the cross-validation set, this effect can hardly be observed (compare Figure A.2 in Appendix A). This again suggests that a significant mismatch between evaluation set and cross-validation set exists. However, the slope differences between the IDR curves is similar on evaluation and cross-validation sets, and the variations in the word class specific curve are also similar on both sets.

As pointed out earlier, the differences between the IDR curves suggest that the HLM is not well-balanced in itself. In order to compensate these differences, the likelihood distribution needs to be modified. In the following section, we discuss an according method which further extends the likelihood scaling technique that was introduced in this section with the application of λ to HLM.

5.9.4 Weight Balancing within HLM

This section presents our method to adjust the weight balance within HLM by applying optimized scaling factors to individual hierarchy levels. In order to separate the scaling *between* AM and LM and the scalings *within* the LM from each other, we still apply the general scaling parameter λ as defined in Equation 5.25. Hence, for within-HLM scaling, each conditional likelihood expression of the search problem formulation could be scaled with a separate parameter. However, in order to limit the number of additional parameters, we only apply separate scaling factors to main hierarchy levels, but not to sub-levels.

For the considered 4-level HLM, a word class factor λ_K and a concept factor λ_C is introduced. Hence, Equation 5.25 becomes:

$$\mathbf{T}^* = \underset{\mathbf{W},\mathbf{K},\mathbf{C}',\mathbf{C}}{\arg\max} \left[P(\mathbf{X}|\mathbf{W}) [P(\mathbf{W}|\mathbf{K})^{\lambda_K} [P(\mathbf{K}|\mathbf{C}')P(\mathbf{C}'|\mathbf{C})]^{\lambda_C} P(\mathbf{C})]^{\lambda} \right] \tag{5.27}$$

For practical WTNH representations of HLM this means that all transition scores within word class networks are now scaled by $\lambda_K\lambda$, and transitions within concept networks are scaled by $\lambda_C\lambda$ instead of λ.

Experiment

The scaling parameters λ, λ_C and λ_K are jointly optimized on the cross-validation set to maximize the tree node accuracy Acc_n. HLM and speech interpretation system configurations are the same as previously, especially as for Figure 5.7. The optimum values of the scaling parameters yielding maximum Acc_n are $\lambda = 18$, $\lambda_C = 1.25$ and $\lambda_K = 1.5$ (compare Figure A.4 in Appendix A).

Figure 5.8 depicts the comparison between the baseline parameter setup with only a common LM factor ($\lambda_C = 1.0$, $\lambda_K = 1.0$) and the optimized parameter setup with individual scaling factors for word classes and concepts ($\lambda_C = 1.25$ and $\lambda_K = 1.5$) on the evaluation set. As in the previous section, the figure shows tree node accuracy (upper plot) and *IDR* (lower plot) curves for the whole hierarchy as well as for individual hierarchy levels, in dependency of λ. The result curves of the baseline are identical to those of Figure 5.7, except for the range of λ. However, the highlighted values of Acc_n and *IDR* here correspond the optimum settings of λ on the cross-validation set, i.e. $\lambda = 21$ for the baseline and $\lambda = 18$ for the balanced HLM.

As can be seen from the upper plot of Figure 5.8, the within-HLM scaling yields accuracy gains on the concept and word class levels almost over the whole tested range of λ. At the optimum setting from cross-validation, the concept accuracy increases from 89.0% to 89.3%, which corresponds to a slight relative error rate reduction of $Err_{rel} = -2.7\%$. On the word class level, a substantial improvement in accuracy from 94.8% to 95.5% can be observed ($Err_{rel} = -13.5\%$). As the word level accuracy decreases slightly ($Err_{rel} = +1.3\%$) and the average number of words is much larger than concepts and word classes together, the total accuracy remains unchanged.

The *IDR* curves corresponding to the two parameter setups are depicted in the lower plot of Figure 5.8. Whereas the total *IDR* value remains near the optimum of 1, the differences between the curves of individual hierarchy levels improve significantly. Especially at the word class level a fundamental improvement both of the whole *IDR* curve and at the optimum λ value (from 7.0 to 1.6) is observed, even if its vertical offset still leaves room for improvement.

On the cross-validation set, within-HLM scaling yields slight accuracy gains on all hierarchy levels (compare Figure A.3). The total accuracy improves from 86.6% to 86.8% ($Err_{rel} = -1.5\%$). Interestingly, the word class *IDR* deteriorates from 0.8 to 0.3 at the points of maximum accuracy.

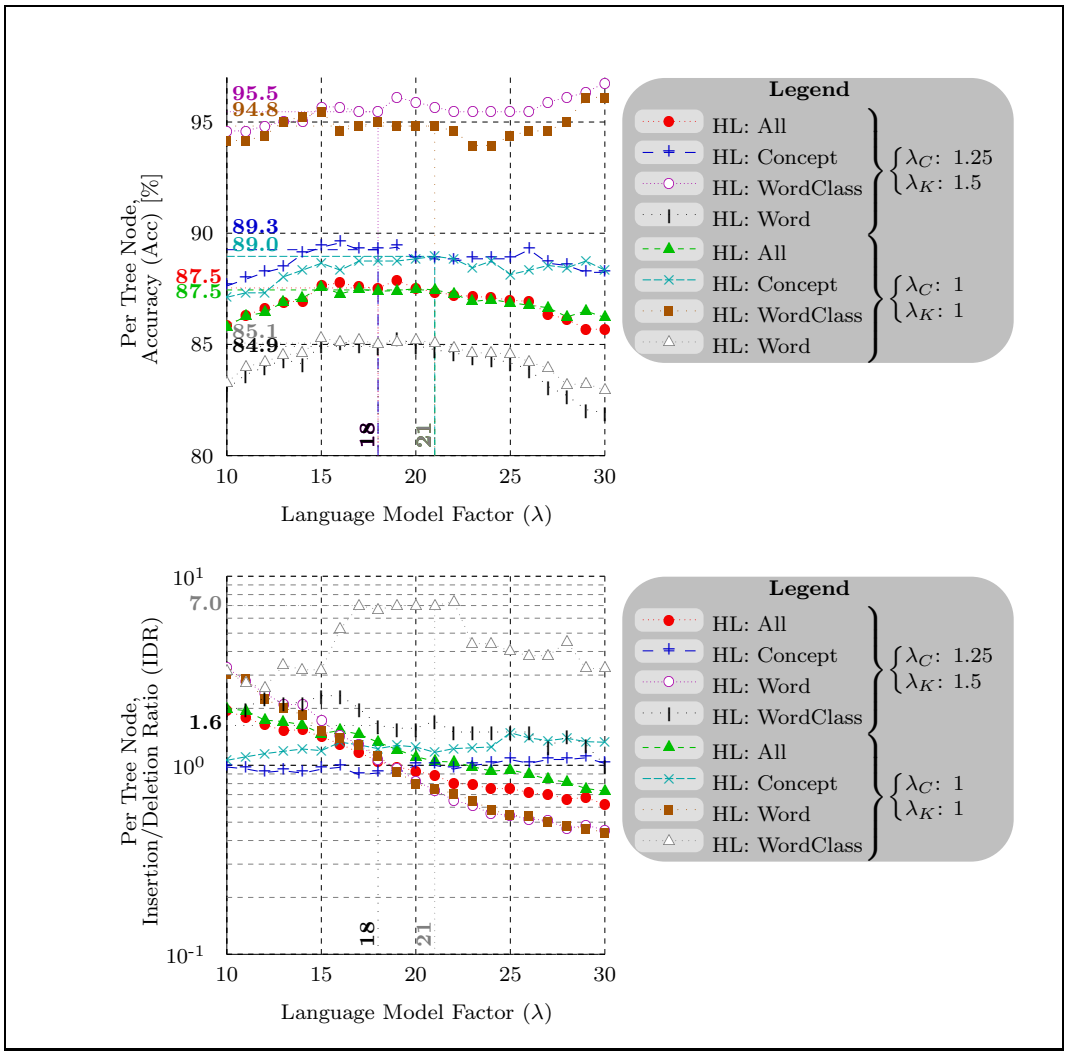


Figure 5.8: *Top: Tree node accuracy with and without level-dependent scaling factors on the evaluation set. Bottom: Insertion-deletion ratio for the identical setup.*

5.9.5 Word Insertion Penalty

As the previous section shows, the within-HLM weight scaling still doesn't yield optimally shaped *IDR* curves. An explanation for this could be that the scalings performed so far are not powerful enough to cope with all requirements, i.e.

- range adaptation of likelihood values,
- balancing of relative influence of AM and HLM,
- balancing of weights within HLM and
- balancing of average word length, i.e. numbers of insertions and deletions.

Especially the latter task is only implicitly influenced by the used scaling factors. Hence, in order to further optimize HLM with respect to their *IDR* curves, the scaling can be extended with a parameter that directly influences *IDR*. This can be achieved with a technique known from speech recognition, which modifies the cost of word transitions by a so-called word insertion penalty [YEK+02].

This parameter enhances the log-likelihood scaling by applying an additive offset p_W to all words. Based on the logarithmic version of Equation 5.27, the search problem can thus be reformulated by including the word insertion penalty p_W:

$$\mathbf{T}^* = \underset{\mathbf{W},\mathbf{K},\mathbf{C}',\mathbf{C}}{\arg\max} \Big[\begin{aligned} &\log(P(\mathbf{X}|\mathbf{W})) + |\mathbf{W}|p_W + \lambda_K \lambda \log(P(\mathbf{W}|\mathbf{K})) \\ &+\lambda_C \lambda \log(P(\mathbf{K}|\mathbf{C}')) + \lambda_C \lambda \log(P(\mathbf{C}'|\mathbf{C})) + \lambda \log(P(\mathbf{C})) \Big] \end{aligned}$$

In the weighted transition network representation of HLM, the word penalization is applied by adding p_W to the score on each transition that leads into a word sub-network node.

Experiment

In order to confirm the effectiveness of p_W together with level-dependent scaling factors experimentally, a joint optimization of λ, λ_C, λ_K and p_W with respect to Acc_n is carried out on the cross-validation set. The maximum of Acc_n is found at parameter settings of $\lambda = 18$, $\lambda_C = 1.25$, $\lambda_K = 1.5$ and $p_W = -10$ (compare Figure A.5). Figure 5.9 depicts the resulting total and per-level tree node accuracy and *IDR* curves on the evaluation set in dependency of λ. In order to measure the effects of p_W alone, the baseline parameter setting corresponds to the optimum setting with level-dependent scaling factors from the previous section, i.e. $\lambda = 18$, $\lambda_C = 1.25$, $\lambda_K = 1.5$ and $p_W = 0$.

The upper plot of Figure 5.9 shows that the use of p_W leads to tree node accuracy gains on the word level (from 84.9% to 85.2%) and on the concept level (from 89.3% to 89.6%). The word class level accuracy remains at 95.5%. In total, we notice an accuracy improvement from 87.5% to 87.8%, which corresponds to a relative error rate reduction of $Err_{rel} = -2.4\%$.

From the lower plot of Figure 5.9 it can be observed that, surprisingly, this accuracy gain occurs although the *IDR* curves don't improve. The word level *IDR* is even slightly further away from the expected optimum value of 1 than the respective baseline value (0.8 vs. 1.1), and the word class *IDR* remains unchanged at 1.6.

Similar effects are observed on the cross-validation set (compare Figure A.6). There, the total accuracy Acc_n rises from 86.8% to 87.2%, which corresponds to $Err_{rel} = -3.0\%$. At the same time, the total *IDR* value at maximum Acc_n worsens from 0.8 to 0.6.

Summary

The tree node accuracy and relative error rate reductions achieved by within-HLM weight scaling and word penalization are summarized in Table 5.2. The upper

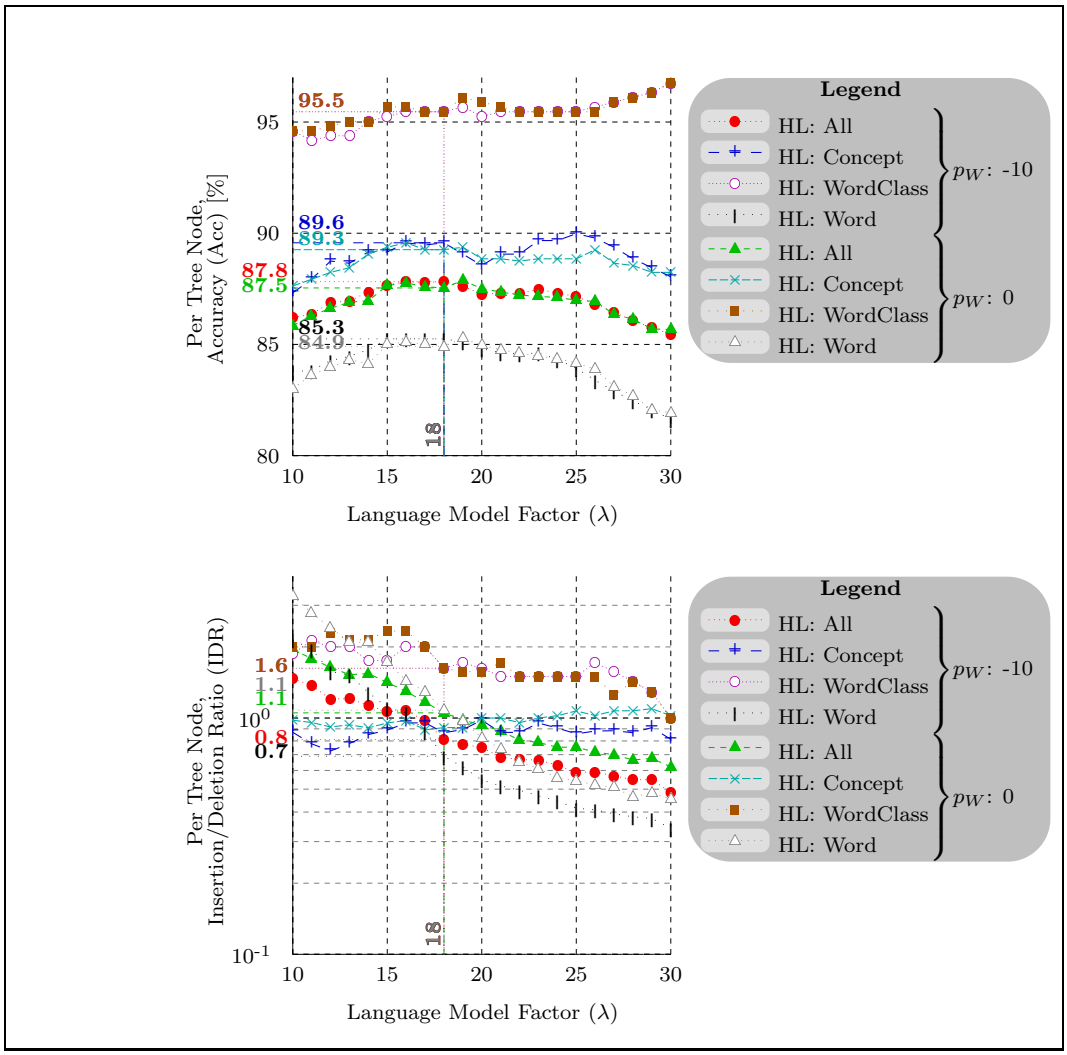


Figure 5.9: *Top: Tree node accuracy with and without word insertion penalty on the evaluation set. Bottom: Insertion-deletion ratio for the identical setup.*

part of this table shows the results of joint parameter optimizations on the cross-validation set. Using these parameter settings on the evaluation set yields the lower part of the table. The baseline column contains the results from only applying a general LM factor λ to HLM (compare Section 5.9.3). The second and third result columns show the values corresponding to optimizations of the within-HLM scaling factors λ_K and λ_C in addition to λ (compare Section 5.9.4). Finally, the results for additionally applying a word insertion penalty p_W to HLM are given in the last two columns of Table 5.2 (compare current section). The reported relative error rate reductions always refer to the baseline values.

set	hierarchy level	baseline	λ_K, λ_C opt.		λ_K, λ_C, p_W opt.	
		Acc_n	Acc_n	Err_{rel}	Acc_n	Err_{rel}
xval	concept	85.5%	85.8%	−2.1%	86.3%	−5.5%
	word class	97.9%	98.1%	−9.5%	98.3%	−19.0%
	word	84.3%	84.5%	−1.3%	85.0%	−4.5%
	total	86.6%	86.8%	−1.5%	87.2%	−4.5%
eval	concept	89.0%	89.3%	−2.7%	89.6%	−5.5%
	word class	94.8%	95.5%	−13.5%	95.5%	−13.5%
	word	85.1%	84.9%	+1.3%	85.3%	−1.3%
	total	87.5%	87.5%	−0.0%	87.8%	−2.4%

Table 5.2: *Summary of results for optimized within-HLM weight scaling and word insertion penalty.*

5.10 Recognition vs. Interpretation

The one-stage speech interpretation system investigated in this thesis can be viewed as an extension of a conventional speech recognition system. This extension consists of the additional semantic knowledge contained in HLM. In this section, we pursue the question how this new knowledge source affects the speech recognition accuracy, i.e. the word accuracy of the system. As outlined in the introduction of this work, the original task of a speech interpretation system is the recognition of meaning. For this task, the creation of a perfect word transcription is not necessary. Nevertheless, corresponding evaluation results are reported in this section, as many scientists attribute importance to them.

Ideally, we would expect an increase in word accuracy from the inclusion of semantic knowledge. This expectation is founded on observations of human speech processing: When humans understand the meaning of what their communication partners say, it is generally easier for them to recognize the uttered words. This observation is e.g. utilized by the semantically-unpredictable sentence (SUS) test [BGH96]. This test is used in speech synthesis research to assess the intelligibility of text-to-speech systems. The test measures the word transcription capabilities of human subjects on synthesized speech. In order to avoid that subjects guess words by using semantic knowledge, the test sentences use common syntactic structures that are randomly filled with words from special word list, e.g. 'The chair ran through the yellow trust.'.

In terms of our speech interpretation system this means that we would expect higher word accuracy from HLM based on semantic concepts than from word class or word based HLM. We answer this question experimentally, by comparing a speech interpretation system and a speech recognition system trained on identical knowledge sources except for the semantic knowledge.

The basic system setup for this experiment is depicted in Figure 5.10. Both

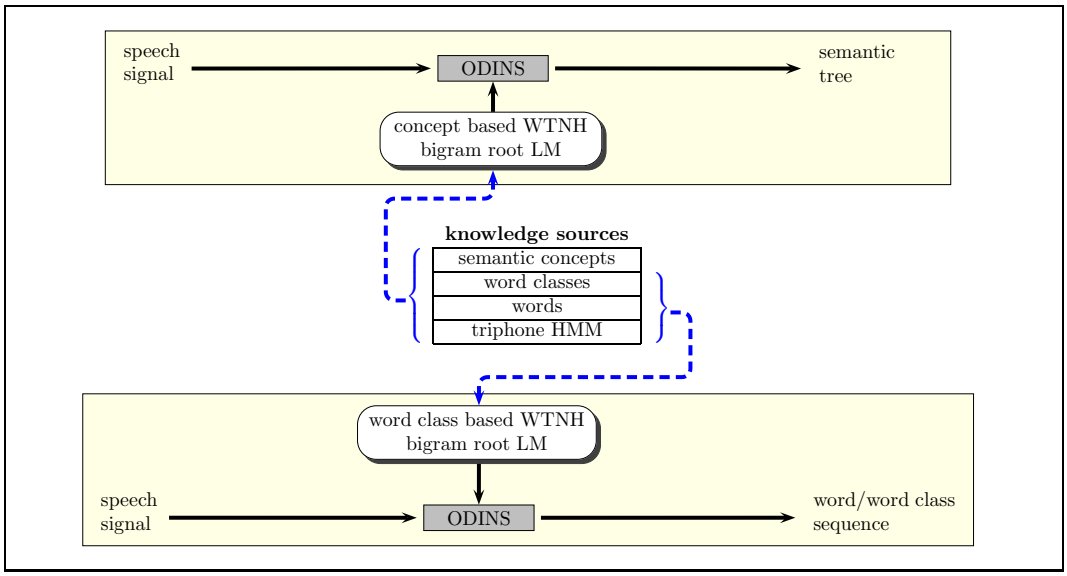


Figure 5.10: *Experimental setup for comparison of speech interpretation (top) and speech recognition (bottom) systems.*

systems rely on identical acoustic-phonetic and lexical models, and both HLM are trained from the tree annotations of the airport information system corpus. For the speech interpretation system (upper part of Figure 5.10) the full annotations are used, whereas the speech recognition system (lower part of Figure 5.10) is trained on the same annotations after hiding the semantic concept levels.

Strictly speaking, the word class level contains semantic information, which doesn't belong to a conventional speech recognition system. Yet, similar as in Section 3.3, omitting the word class models would result in a smaller vocabulary (compare Section 5.6.1) and thus render the comparison unfair. Therefore, word classes are considered a part of the speech recognition system.

Evaluation is carried out by semantic tree matching for both systems, since the output sequences of words and word classes of the speech recognition setup (compare Figure 5.10) are also represented as trees. In addition to tree node accuracy, the test-set perplexity values $ppl(\mathbf{T})$ for the HLM of the two system are reported. Although only one of them contains semantic concepts, the two HLM are comparable due to the word based normalization of $ppl(\mathbf{T})$ (see Section 4.4). In order to enable a tree node accuracy based comparison, the evaluation of the speech interpretation system is only performed after deleting all semantic concepts from the decoder hypotheses and from the reference trees. Thereby, only the word and word class levels of both systems affect the evaluation. This also ensures that the parameter optimization is, in both cases, carried out with respect to the joint accuracy of words and word classes, represented by the total tree node accuracy Acc_n.

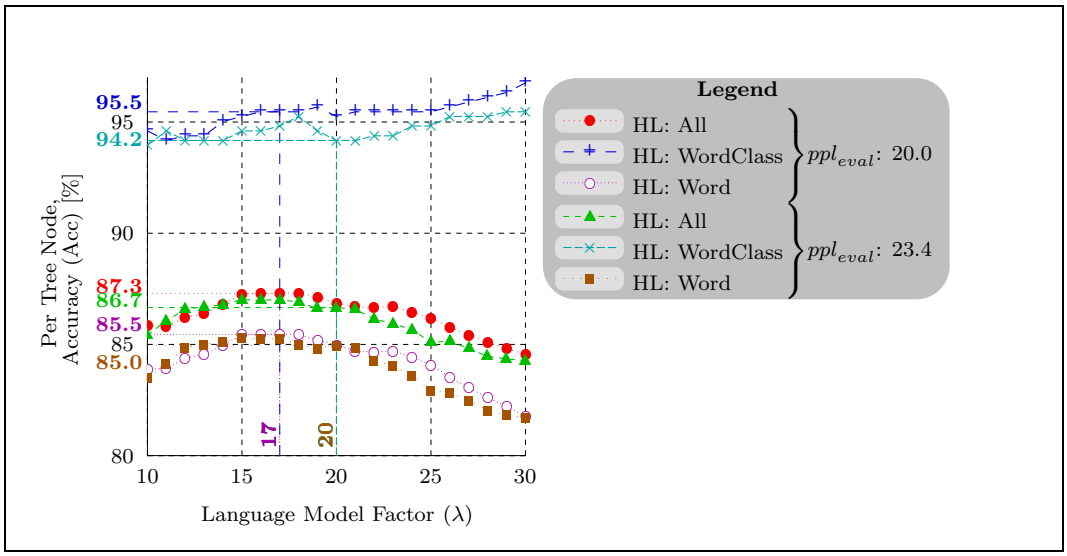


Figure 5.11: *Total and per-level tree node accuracy for speech interpretation (upper 3 curves of legend) and recognition (lower 3 curves) systems on the evaluation set.*

Please note that, as already pointed out in Section 4.3.2, the word level accuracy Acc_n^w reported during this evaluation is generally different from an isolated evaluation of the word sequence alone (which would yield Acc_w), because of the constraints imposed by the word class identities. In the particular experiment regarded here, however, the differences are marginal.

The standard speech interpretation system configuration elaborated in previous sections is used for this experiment, i.e. a 4-level HLM with bigram root LM, modified Kneser-Ney smoothing and Good-Turing discounting. HLM scaling and word penalization are applied according to the previous section. The parameter settings that maximize the joint word and word class accuracy Acc_n are $\lambda = 17$, $\lambda_C = 1.25$, $\lambda_K = 1.5$ and $p_W = -10$ (compare Figure A.7). The word class based, 2-level HLM for the speech recognition system is built with the same techniques as the concept based HLM, i.e. with bigram root LM, modified Kneser-Ney smoothing and Good-Turing discounting. Likewise, HLM scaling and word penalization are applied. The joint parameter optimization on the cross-validation set leads to settings of $\lambda = 20$, $\lambda_K = 1.5$ and $p_W = -10$ (compare Figure A.8).

Figure 5.11 depicts the tree node accuracy curves of both systems for all and individual hierarchy levels on the evaluation set. The numerically highlighted data points correspond to the settings of maximum total tree node accuracy, i.e. $\lambda = 17$ for the concept based HLM and $\lambda = 20$ for the word class based HLM. On the word level as well as on the word class level, the use of semantic knowledge results in accuracy improvements. The total accuracy cannot be compared in this experiment because it considers semantic concepts, which are only contained in the HLM of the

speech interpretation system.

On the word level, we notice an accuracy improvement from 85.0% to 85.5% for the concept based HLM. This corresponds to a relative error rate reduction of $Err_{rel} = -3.3\%$. The word class level yields a more substantial accuracy gain from 94.2% to 95.5% ($Err_{rel} = -22.4\%$). The joint accuracy for both levels improves significantly from 86.7% to 87.3% ($Err_{rel} = -4.5\%$). On the cross-validation set, error rate reductions of $Err_{rel} = -3.9\%$, $Err_{rel} = -60.5\%$ and $Err_{rel} = -6.8\%$ are obtained for words, word classes and both together, respectively (compare Figure A.9). The test-set perplexity values of the two HLM, which are shown in the legend of the figures, improve from 23.4 to 20.0 on the evaluation set and from 21.8 to 19.8 on the cross-validation set. The presented results confirm our expectation that the use of semantic knowledge has a positive effect on speech recognition accuracy.

Chapter 6

Unknown Word Modeling

In closed-vocabulary speech recognition and understanding systems, users are expected to utter only those words that are contained in the system lexicon. Therefore, these types of systems take no explicit provisions for the case that unknown words occur. These words are also called *out-of-vocabulary* (OOV) words, whereas words contained in the system lexicon are denoted as *in-vocabulary* (IV) words. Occurrences of OOV words inevitably cause errors in closed-vocabulary systems, because they cannot be distinguished from IV words and are therefore silently misrecognized as IV words. Even worse, surrounding words are often also affected by these errors, as a consequence of erroneous segmentations of the misrecognized OOV words and also due to erroneous language model contexts. In speech recognition experiments reported in [Fet98], each OOV word caused about two misrecognized words on average. The usefulness of a closed-vocabulary approach for a particular type of application therefore depends on the relative number of OOV words (also called OOV rate) that occur during system operation, and if the amount of errors caused by these OOV words is acceptable. The one-stage speech interpretation system discussed in this work is so far configured as a closed-vocabulary system. Later on in this chapter, we give reasons why we expect high rates of OOV words during practical operation of our type of system. Therefore, the unknown word problem is an important issue in this work.

This chapter discusses methods to alleviate this problem by adding explicit knowledge of unknown words to the system. Thereby, ODINS becomes an open-vocabulary system that is able to detect words not contained in the system lexicon. As further elaborated in Section 6.1, explicit OOV models are in principle not only able to detect OOV words, but also to avoid interference with their surroundings. This section also discusses which processing levels an OOV model can be based on, and why a lexical level approach is chosen in this work. Our OOV modeling approach, which is presented in Section 6.2, reuses data-driven language modeling techniques from Chapter 5 to estimate statistical LM over phonemes. Large pronunciation lexica and, optionally, word frequency lists are used as knowledge bases. The main features of this approach are also published in [TFLR05b]. Section 6.3

then shows how these unknown word models are integrated into HLM and into the decoding process of ODINS. Suitable methods to measure the detection capabilities of OOV models and their effects on semantic accuracy are outlined in Section 6.4. Finally, Section 6.5 discusses experimental results for various system configurations.

6.1 Motivation and Basic Approach

As mentioned in the introduction, the need for OOV modeling also depends on the OOV rate occurring in a particular application. For large-vocabulary dictation applications, closed-vocabulary systems are often appropriate because OOV rates are typically low. Due to the difficulty of the task of speech understanding, these systems today mostly operate in narrow application domains with limited vocabulary sizes of a few hundred or thousand words. In these circumstances, high rates of OOV words must be expected if users talk freely to the system. This can even be the case if all semantically relevant words are known to the system, when users are oblivious to domain limitations.

The airport information corpus presented in Section 4.1 yields a relatively low OOV rate of about 2% on the test sets (compare Table 4.2). While the error rate increase caused by this OOV rate itself may be acceptable, there are reasons why we expect substantially higher OOV rates in real-world situations:

- Although the speech corpus is collected through Wizard-of-Oz (WOZ) experiments which simulate real-world situations (compare Section 4.1.1), the subjects need to be instructed about the kind of queries they should make. Naturally, this tempts the subjects to reuse words and phrases occurring in the instructions. Hence, a more natural choice of words and therefore a higher OOV rate can be expected in real-world system use.
- Since real-world users can be expected to be less aware of domain limitations than WOZ subjects, a larger number of queries can be expected whose answering requires knowledge unknown to the system. Such queries are likely to contain a large proportion of unknown words.

There are also other reasons against a closed-vocabulary approach:

- For a spoken dialogue application, the ability to detect OOV words at all can be vital in order to purposefully ask users to rephrase parts of their utterance, instead of continuing with the misrecognized words.
- It may be desirable to obtain a system vocabulary which only, or mainly, contains words that are semantically relevant for the application. In the extreme case, the speech interpretation system would thereby become similar to a keyword spotting system. One way to exclude semantically irrelevant words from the system lexicon is to declare them as unknown. In this work, this is achieved by replacing their occurrences in the corpus annotations by the unknown word symbol (compare Section 6.3). Thereby, the 'natural' OOV rate and hence the need for OOV modeling is increased. In contrast to the 'natural' OOV

words that never occur in the training data, we denote those words that we deliberately declare unknown as *known OOV words*.

In the following, we present a brief categorization of existing statistical OOV modeling approaches. For a more detailed survey, the interested reader is e.g. referred to [Fet98]. In general, the methods for detecting unknown words can be divided into implicit and explicit techniques. In an implicit approach, the speech decoding process typically does not differ from that of a closed-vocabulary system. Instead, OOV words are detected by using knowledge collected during the normal search process. As described in [Fet98], implicit OOV detection can be approached by confidence measurement. Confidence measures represent estimates about the certainty of decoder hypotheses, e.g. on a word-by-word basis. Ideally, the confidence value is 0 for erroneous word hypotheses and 1 for correct ones. Since OOV words cause word errors, they can in principle be detected by use of confidence measures. However, low confidence values do not exclusively result from unknown words, but generally from bad matches of the input signal on acoustic-phonetic, lexical or language modeling levels. Therefore, it is impossible to unambiguously identify OOV words by this means. Moreover, implicit OOV modeling is unable to avoid errors caused by misrecognized OOV words. In contrast to this, explicit approaches are able to both detect OOV words and avoid these types of errors. For this purpose, the OOV model is directly integrated into the decoding process. On the downside, an explicit OOV model effectively introduces new word pronunciations into the system, which generally increases confusability. Hence, care must be taken that the negative effects of the OOV model don't outweigh its benefits.

Explicit, statistical OOV models are typically based on the acoustic-phonetic or on the lexical modeling level. The acoustic-level OOV model introduced in [Fet98] consists of several HMM for OOV words of different length. This requires that sufficient amounts of acoustic data are available for HMM training as additional knowledge source. Lexical-level OOV models combine existing phoneme HMM in a suitable way to yield one or several generic pronunciation models. For this purpose, pronunciation lexica can be utilized as an additional knowledge source, as e.g. proposed by Bazzi [Baz02].

For the speech interpretation task pursued in this work, OOV modeling on the lexical level is considered more appropriate than on the acoustic-phonetic level. Because of the above mentioned method of defining known OOV words, it is assumed that the unknown words are mostly common words of the target language. Therefore, they aren't expected to have substantially different acoustic-phonetic properties than IV words. Hence, we see little reason, regarding model goodness, to base OOV words on that processing level. A practical argument for lexical OOV modeling is the availability of large pronunciation lexica for many languages, whereas it may be more difficult to obtain speech data for acoustic-phonetic OOV model training. Finally, as discussed in [Fet98], the main drawbacks of lexical OOV modeling are a tendency for over-generation and the resulting need for OOV model penalization,

and high computational requirements. Whereas the former issue is addressed in this chapter by examining the sensitivity of OOV models against penalty variations, analysis of runtime performance is not a focus of this work. Nonetheless, the effects of reducing OOV model sizes on model goodness are also examined in Section 6.5.

6.2 Lexical OOV Modeling with Statistical LM

As outlined in the previous section, it is expected that most of the unknown words are common words from the language of the target application, in our case German. Consequently, we represent OOV words by a generic pronunciation model for arbitrary words. In order to estimate this model, pronunciation lexica are used as primary knowledge source. A statistical OOV word model can then be generated by use of statistical language modeling techniques, only that the described symbol sequences now consist of phonemes instead of words or semantic categories, as was the case in Section 5.3.

A similar approach to OOV word modeling was proposed by Bazzi [Baz02]. Yet, the methods presented here differ in several basic aspects: Bazzi applies his OOV models to a WFST-based speech recognition system, which is part of a multi-stage speech understanding system. The speech recognizer makes use of the finite-state transducer technique outlined in Section 2.2.2. In contrast, the one-stage speech interpretation system utilized here is based on an explicitly hierarchical automata representation in the shape of WTNH. Due to the one-stage approach, our OOV model directly affects the whole syntactic-semantic processing. Both systems have in common that the OOV word model representation is compacted by automata minimization. With regard to the OOV model itself, Bazzi primarily concentrates on automatic creation of suitable variable-length subword units and on modeling multiple classes of OOV words. Here, we focus on investigating the statistical phoneme LM itself by applying two different language modeling techniques, and we examine model performances for different OOV rates.

Two different methods for statistical language modeling were outlined in Section 5.3, namely n-gram LM and exact LM. There, these techniques are utilized to model the likelihood of sequences of words and semantic category symbols within HLM. In this chapter, we reuse these methods to describe the likelihood of phoneme sequences produced by the generic pronunciation model for OOV words. Instead of training the statistical LM on semantic symbol sub-sequences from the tree annotation corpus, the training set here consists of the phoneme sequences of word pronunciations from a pronunciation lexicon. As in Section 5.3, we use smoothing and discounting to adjust the likelihood distribution of phoneme LM, and we also represent the resulting LM as weighted transition networks in order to integrate them into WTNH.

Exact phoneme LM exactly cover the pronunciations occurring in the training set. Hence, they can be viewed as 'super-words', which combine different word pro-

nunciations in a single lexical item. By means of their weight distribution, exact phoneme LM assign likelihood values to pronunciations proportional to their occurrence frequencies. In contrast to exact phoneme LM, n-gram phoneme LM have limited context dependency, and therefore also cover unseen phoneme sequences. This property may be desirable in order to broaden the OOV word model's coverage of pronunciation variations. This especially includes cut-off words, which can occur frequently in natural speech (compare Section 5.5).

As discussed in Section 5.3.1, the generalization capabilities of n-gram LM can be extended by combination with lower-order models. Yet, in the case of phoneme n-gram LM it may be advantageous to limit generalization, in order to prevent over-generation of pronunciations and thereby confusions with IV word pronunciations. Furthermore, since the phoneme models of our test system are triphone HMM, it needs to be ensured that these are traversed in the correct order, i.e. with matching left and right contexts. For those reasons, we disallow unigram backoff altogether, and examine how inclusion of lower-order backoff affects the performance of phoneme n-gram LM. Backoff to lower-order n-grams can be removed from the transition network representation of n-gram LM by deleting the concerned failure transitions (compare Section 5.3.1).

Two different German pronunciation lexica are utilized as knowledge sources for phoneme LM estimation, namely Phonolex [pho04] and Celex [BPG95]. From Phonolex, the manually verified 'core' pronunciations of 22k inflected words were used. As the phoneme sets of Phonolex and ODINS are both similar to the Verbmobil definitions (compare Section 5.6.2), only few phoneme mappings need to be performed. The German Celex database contains 52k lemmata with 366k corresponding wordforms. The adaptation of its phoneme set is more difficult. Wordforms containing foreign phonemes or others which have no well-defined counterpart in our phoneme set are removed. After this, 314k pronunciations remain for OOV model estimation.

In addition to word pronunciations Celex also contains word frequency information from different sources, including spontaneous speech transcriptions. This information is optionally used to weight pronunciations accordingly. This is achieved by modifying the pronunciation list that is used to train phoneme LM. The modification copies each word pronunciation as many times as the word frequency value suggests, and thereby simulates a corpus with the desired frequency statistics.

As mentioned in the previous section, integrating an explicit OOV model into the decoding process increases confusability. In order to lower confusability between OOV and IV words, we examine the effects of excluding IV word pronunciations from the training set for phoneme LM.

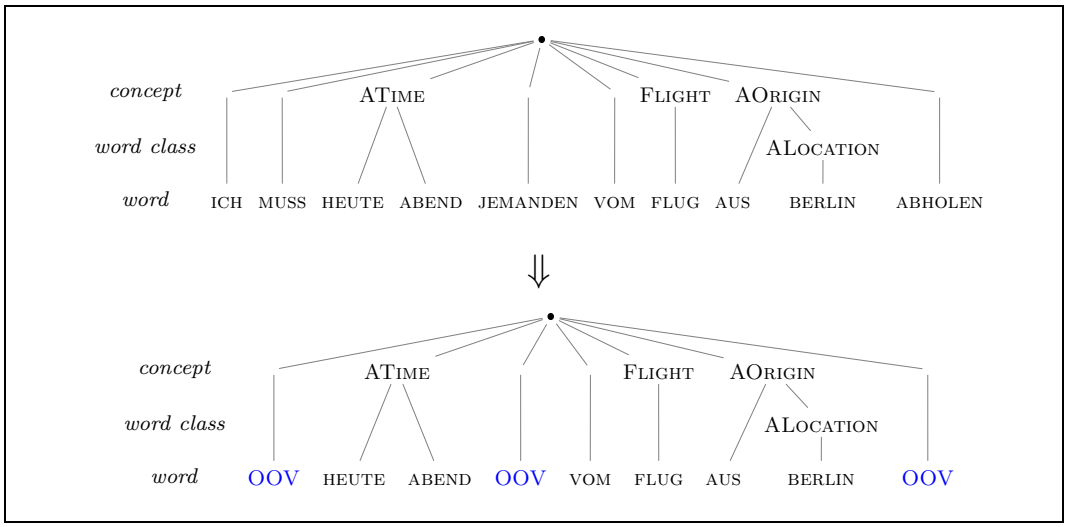


Figure 6.1: *Replacing known unknown words with OOV symbols in semantic tree annotations.*

6.3 Integration into WTNH

In order to integrate an OOV word model into the decoding process, it needs to be defined at what positions OOV words may occur in an utterance and which likelihood is assigned to them. Hence, the OOV word model must be considered by the language model, in our case by the HLM. This can be carried out transparently, i.e. without the need to modify the HLM building process from Chapter 5, if OOV words become part of the tree annotations of the training corpus.

For this purpose, some of the words previously annotated as known now have to be declared as unknown. While this may not be a suitable approach for a large-vocabulary dictation application, where all words seen during training are included in the system vocabulary, it may be desirable for a small- to medium-vocabulary dialogue scenario to exclude semantically irrelevant words from the system lexicon, as outlined in Section 6.1. In order to achieve this, the existing semantic symbol set is augmented with the new word symbol 'OOV'. Then, the tree annotations are modified by replacing the symbols of semantically irrelevant words by the OOV word symbol. As mentioned before, these words are denoted as *known OOV words*, in contrast to the 'natural' OOV words, i.e. those words of the test sets that are never encountered in the training set.

Figure 6.1 shows an example annotation tree from the airport information corpus for the utterance ICH MUSS HEUTE ABEND JEMANDEN VOM FLUG AUS BERLIN ABHOLEN (word-by-word translation: I must today evening someone from flight from berlin collect). In the lower annotation tree of Figure 6.1, the words ICH MUSS, JEMANDEN and ABHOLEN are declared as unknown by replacing them with the OOV symbol.

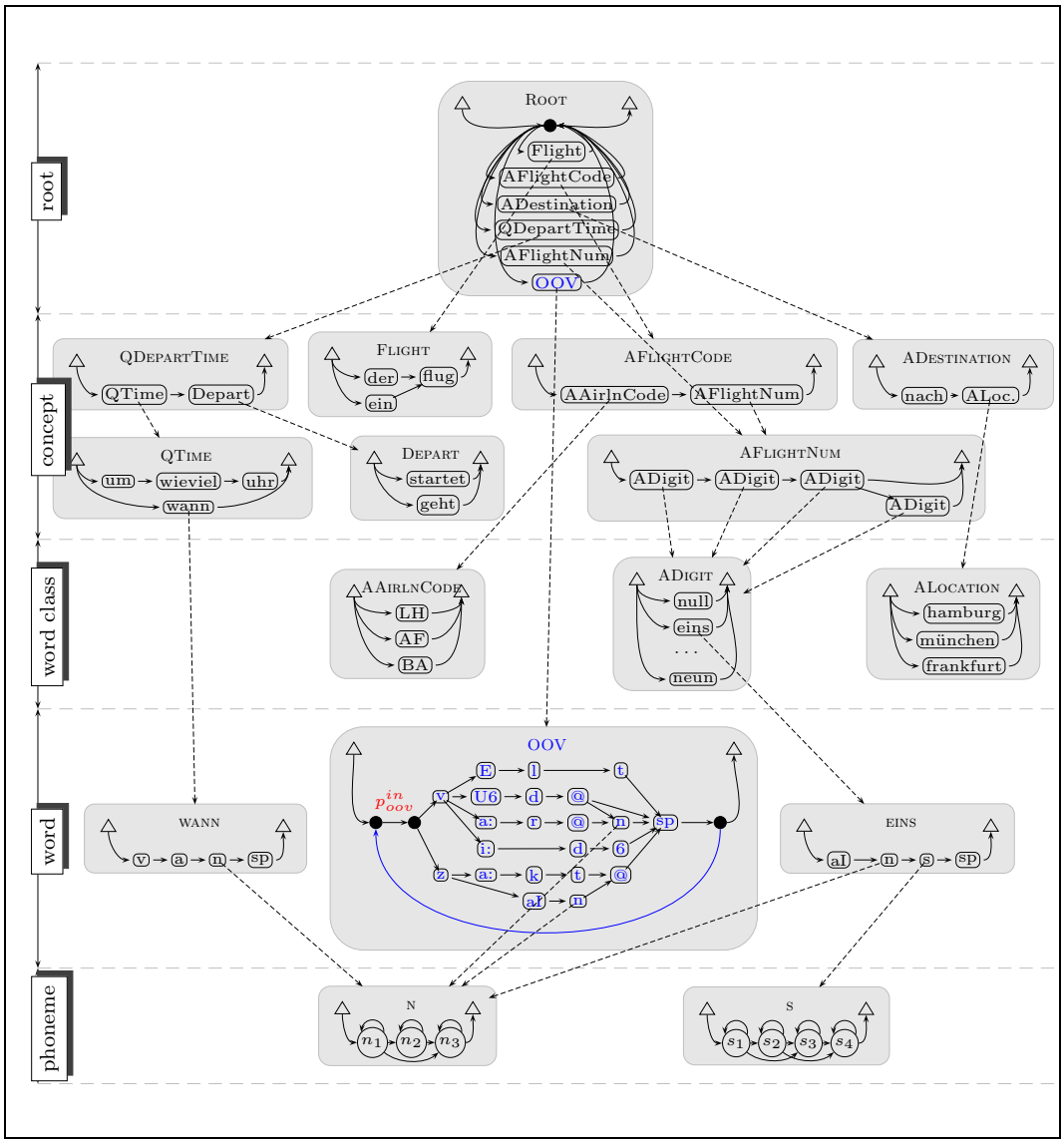


Figure 6.2: *WTNH of Figure 2.2 with integrated lexical OOV word model.*

Generally, candidates for known OOV words are all semantically irrelevant words within the given application domain. It may nevertheless be desirable to keep some of the semantically irrelevant words in the vocabulary, because they have syntactic relevance or because their effect on confusability is larger as OOV word than as IV word. In this work, we examine two different sets of OOV word annotations on the airport information corpus, yielding total OOV rates of about 8% and 23% on the evaluation set, respectively. In the so-called 23%-set, all surface words, i.e. all words not contained in a semantic category, are declared as unknown. In contrast, some

of the most frequent syntactically relevant surface words are not declared unknown in the so-called 8%-set.

Please note that we combine consecutive OOV words, e.g. ICH MUSS in Figure 6.1, into a single OOV symbol (but still compute OOV rates in terms of individual OOV words). This measure enables a more robust estimation of OOV model likelihood within HLM, because it avoids context dependencies between OOV words, and reduces the required LM dependency range across OOV words. Furthermore, the OOV model is only required to correctly detect occurrences of OOV words for this thesis, but not *how many* OOV words were uttered consecutively, and how the speech signal is segmented into them. We motivate this simplification with observations of humans, since even they have difficulties to correctly segment sequences of spoken words that are unknown to them. As a consequence, consecutive OOV word sequences are also used as basic unit for OOV model evaluation. For simplicity, we still denote consecutive OOV word sequences as OOV words in the following.

Finally, the network representation of the generated OOV word pronunciation model is included in the WTNH. Based on the example of Figure 2.2, a WTNH with integrated OOV model is depicted in Figure 6.2. In this example, the OOV word symbol is only contained in the root LM of the HLM. The OOV network itself can produce consecutive sequences of 'real' OOV words by following the edge from the null node that precedes the exit node back to the null node which follows the entry node.

In order to balance the relative weighting of IV and OOV words, penalization of OOV models is carried out with two different parameters. An additive log-likelihood offset p_{oov}^{in}, also called OOV entry penalty, is applied when entering the OOV word model. Figure 6.2 displays this parameter at the transition between the two null nodes after the entry node. A log-likelihood scaling factor denoted λ_{oov}, which is not shown in the figure, is applied multiplicatively to all transition scores within the OOV network, more precisely after p_{oov}^{in}. These two parameters are similar to the scaling and offset parameters λ and p_W for relative weighting of acoustic-phonetic and language models, as discussed in Section 5.9.

6.4 Evaluation Method

The unknown word problem can be viewed as the task of detecting OOV words within sequences of IV words. Within this context, the detection of an OOV word is denoted as an *acceptance* (of an OOV word), whereas the classification as an IV word is denoted as a *rejection* (of an OOV word). Detection tasks are typically evaluated by plotting various operating points into a so-called *receiver-operating-characteristic* (ROC) diagram. Such a diagram shows the two types of errors that the OOV word detector can make, namely false acceptances (*FA*) and false rejections (*FR*). *FA* denotes an IV word wrongly hypothesized as OOV word, *FR* an OOV word wrongly hypothesized as IV word. The two types of correct operations are called correct acceptance (*CA*) and correct rejection (*CR*). Table 6.1 gives an overview over the four

Reference	Hypothesis	Detector State
OOV	OOV	correct acceptance (CA)
IV (including ϵ)	OOV	false acceptance (FA)
IV (including ϵ)	IV (including ϵ)	correct rejection (CR)
OOV	IV (including ϵ)	false rejection (FR)

Table 6.1: *Mappings between IV and OOV words for evaluation of OOV detection performance.*

possible mappings between reference transcription and decoder hypothesis with regard to the two classes of symbols (OOV or IV). The counts of FA, FR, CA and CR are computed from the mappings between reference transcriptions and hypotheses of a test set. As discussed in Section 4.3.1, such mappings are computed by semantic tree matching in this work. Since the OOV detection task merely relates to words, only the word level mappings from the tree match are considered. In addition to mappings between OOV words and IV words, insertions and deletions occur. These can be viewed as mappings from or to empty symbols ϵ, respectively. In order to take these kinds of errors into account when evaluating the detection performance of OOV models, ϵ is treated as an IV symbol, as shown in Table 6.1. Hence, insertions of OOV words are counted as false acceptances, OOV word deletions are counted as false rejections and IV word insertions or deletions are counted as correct rejections.

In this work, ROC curves are plots of false acceptance rate (FAR) against false rejection rate (FRR). FAR is defined as the number of FA in relation to all IV words in the reference transcription. Likewise, FRR is defined as the number of FR in relation to all OOV words in the reference transcription:

$$FAR = \frac{FA}{CR + FA} \qquad FRR = \frac{FR}{CA + FR}$$

The left diagram of Figure 6.3 shows the basic appearance and properties of an ROC plot. A typical ROC curve, which always ranges between the data points $(0, 1)$ and $(1, 0)$, has a hyperbolic shape. The more a curve approaches the FAR and FRR axes, the less confusions occur between OOV and IV words. Ideally, both FAR and FAR are zero. The baseline ROC curve, i.e. the curve corresponding to an OOV detector which is randomly guessing, is a straight line between the start and end points.

In order to capture the course of an ROC curve in a single evaluation measure, the area over the curve is computed (see right diagram of Figure 6.3). Such a measure is denoted as *figure-of-merit* (FOM). Its ideal and baseline values are 100% and 50%, respectively. If we denote the ROC curve over the FAR values with $r(x)$, the FOM can be expressed as:

$$FOM = 1 - \int_0^1 r(x)$$

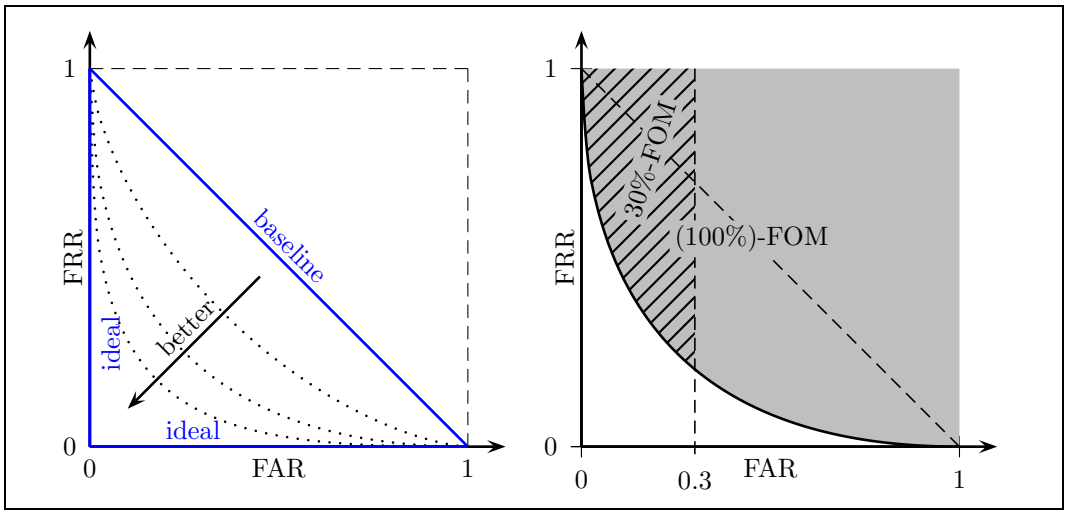


Figure 6.3: *Left: Basic appearance of ROC curves. Right: Figure-of-merit computation.*

For a practical open-vocabulary speech interpretation application, one usually tries to avoid that IV words are misinterpreted as OOV words. Hence, the *FAR* should be small. In order to focus OOV model evaluation accordingly, *FOM* can be limited to this region of interest by computing the area only up to a certain *FAR* limit l. The area is normalized to yield a constant value range between 0 and 1. In this work, we use $l = 5\%$, and denote the corresponding evaluation measure as 5%-*FOM*. Please note that Figure 6.3 shows the 30%-*FOM* for better clarity. Generally, an l-*FOM* is computed by:

$$l\text{-}FOM = 1 - \frac{1}{l}\int_0^l r(x)$$

Note that the baseline value of the l-*FOM* is no longer 50%, but $l/2$.

In order to plot ROC curves, various operating points are determined by running a test set with different settings of the OOV scaling and penalty parameters p_{oov}^{in} and λ_{oov} through ODINS. Low penalty values produce operating points towards the right hand side of the OOV curve, since IV words tend to be recognized mistakenly as OOV words. Increasing the penalties produces operating points towards the left side of the plot. In this work, constant value ranges of $p_{oov}^{in} = -7 \ldots 0$ and $\lambda_{oov} = 0.5 \ldots 20$ were used for all experiments, in order to ensure comparability.

The detection of OOV words also affects the recognition of IV words, since one or several IV words (or parts of IV words) can be mistakenly recognized as OOV words, or vice versa. Therefore, it is essential to include an evaluation of the IV words in an OOV model evaluation. Moreover, as we use the OOV model within a speech interpretation system, the whole semantic representation can be influenced,

so that the semantic tree node accuracy Acc_n (as defined in Section 4.3.2) must be taken into account instead of only the word accuracy. Note that the computation of Acc_n does not consider OOV words, whereas the computation of FAR and FRR does. Hence, we match each pair of semantic trees a second time after deleting the OOV nodes from both reference and hypothesis trees.

For a simultaneous analysis of both evaluation measures, they are combined in a single diagram by using a common abscissa but two different ordinate axes. While the ROC curve (which only considers the word level) is plotted as usual with FAR on the abscissa and FRR on the ordinate, the tree node accuracy (considering all hierarchy levels) is plotted on a second ordinate, which is located on the right side of the diagram. Thereby, the performance of the OOV model with regard to both the whole speech interpretation system and the best operating region can be read off a single diagram.

6.5 Experiments

In this section, the results of a number of experiments with different OOV model configurations are presented. First, we illustrate the effects of varying the penalization parameters of OOV models, and show a combined ROC-accuracy plot for two different phoneme language modeling methods. Then, the performance of different types of phoneme LM with different smoothing techniques is investigated. Furthermore, we present comparisons between two pronunciation lexica for OOV model training, and examine the influence of frequency weighting and exclusion of IV word pronunciations from OOV training. Finally, the robustness of our OOV modeling approach against different rates of OOV words is tested.

According to Section 4.1.2, the 'natural' OOV rate on the airport information corpus is 2.2% on the evaluation set and 1.5% on the cross-validation set. As explained in Section 6.3, two sets of OOV word annotations are used to examine how the system performs for different OOV rates. For the so-called 8%-set, the rates of known OOV words are 8.6%, 4.7% and 7.2% on the training, cross-validation and evaluation sets, respectively. The second annotation set yields known OOV word rates of 25.2%, 16.1% and 21.6%, respectively, and is referred to as the 23%-set. From the 620 IV words in the lexicon of the baseline system without OOV modeling, about 400 IV words remain in the 8%-set, and about 380 IV words remain in the 23%-set.

The open-vocabulary speech interpretation system employed for the experiments discussed in this section also uses the basic setup described in Section 5.6. Since the computational costs for varying the OOV model penalization parameters alone are already high, all experiments were carried out with fixed settings of the HLM scaling and penalization parameters. The used settings correspond to the ones yielding maximum Acc_n (on the cross-validation set) for the closed-vocabulary system determined in Section 5.9.5, i.e. $\lambda = 18$, $\lambda_C = 1.25$, $\lambda_K = 1.5$ and $p_W = -10$.

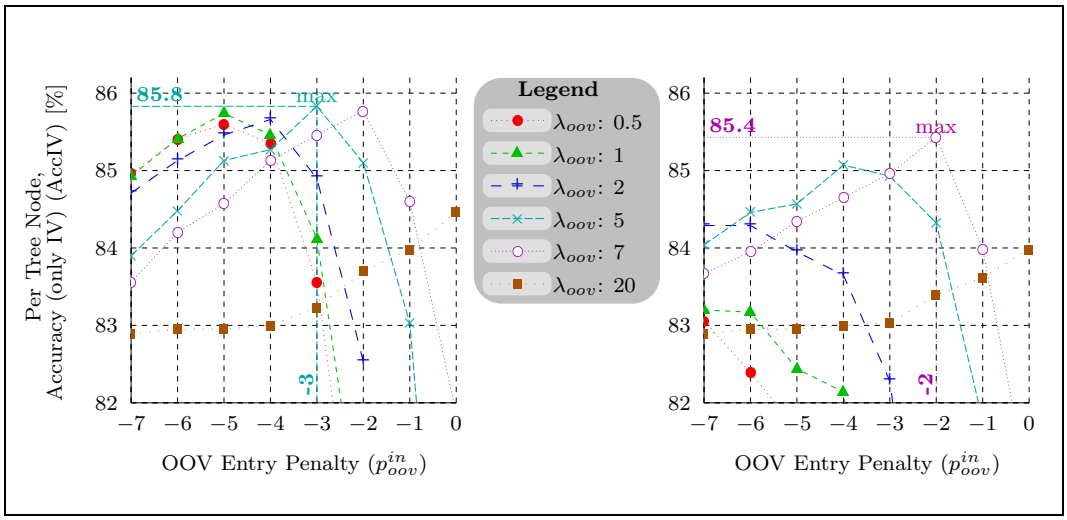


Figure 6.4: *Acc_n for varying OOV penalties p^{in}_{oov} and λ_{oov} with exact (left) and trigram (right) phoneme LM, trained on Phonolex, on cross-validation and 8%-set.*

In all experiments, the OOV model penalties p^{in}_{oov} and λ_{oov} were first optimized on the cross-validation set to yield maximum Acc_n. Those settings were then used to determine OOV model performance on the evaluation set, particularly Acc_n and the values of *FAR* and *FRR* at this operating point. In order to compute *FOM* values, p^{in}_{oov} and λ_{oov} are varied on the evaluation set, too.

6.5.1 Penalization Parameters

Figure 6.4 illustrates tree node accuracy curves for different settings of λ_{oov} in the range $p^{in}_{oov} = -7 \ldots 0$. This experiment is performed on the cross-validation set of the 8% OOV annotation set. For the left diagram, an additively discounted exact phoneme LM is used as OOV model, the right diagram shows the results for a modified Kneser-Ney smoothed trigram phoneme LM. With increasing scaling factor λ_{oov} and decreasing additive offset p^{in}_{oov}, the log-likelihood of the OOV model generally declines in relation to the IV model likelihood. Towards the left side of the diagram, the penalization of p^{in}_{oov} becomes so strong that more OOV words are on average mistakenly recognized as IV words than vice versa, i.e. *FRR* is high and *FAR* is low. The opposite effect occurs towards the right diagram side. The maximum tree node accuracy occurs when the likelihood of OOV and IV models is well-balanced. With rising penalization through λ_{oov}, the requirement for additional penalization by p^{in}_{oov} decreases, and the accuracy maxima move towards the right diagram side. The maximum accuracy of $Acc_n = 85.8\%$ is achieved at $(p^{in}_{oov}, \lambda_{oov}) = (-3, 5)$ for the exact phoneme LM, and $Acc_n = 85.4\%$ at $(p^{in}_{oov}, \lambda_{oov}) = (-2, 7)$ for the trigram phoneme LM.

It can be noted that the accuracy maxima for the exact phoneme LM have

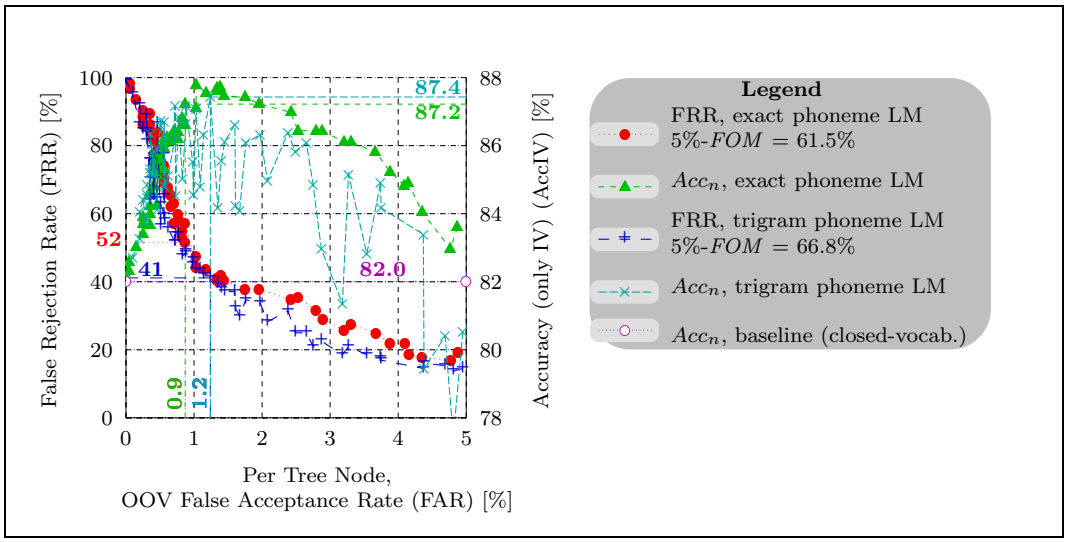


Figure 6.5: *Combined plot of ROC curve and Acc_n for exact and trigram phoneme LM, trained on Phonolex, on evaluation and 8%-set.*

similar values, which suggests that different combinations of p_{oov}^{in} and λ_{oov} achieve similar effects on this type of OOV model. In contrast to this, the maxima for the trigram phoneme LM differ significantly, e.g. 85.0% for the second maximum vs. 85.4% for the first one. Hence, the penalization effects of p_{oov}^{in} and λ_{oov} seem to be not interchangeable on this model type.

6.5.2 Exact vs. n-Gram Phoneme LM

Figure 6.5 depicts the combined ROC-accuracy plot for the same OOV models as used for Figure 6.4, but here for the evaluation set. Please note that the two ordinate axes are scaled differently. As mentioned above, the numerically highlighted values correspond to the settings of p_{oov}^{in} and λ_{oov} yielding maximum Acc_n on the cross-validation set. At these operating points, the OOV model based on an exact phoneme LM has $FAR = 0.9\%$, $FRR = 52\%$ and $Acc_n = 87.2\%$. The 5%-FOM of its ROC curve is 61.5%. The trigram phoneme LM achieves a slightly better accuracy of $Acc_n = 87.4\%$ and a substantially better OOV detection performance both at the operating point ($FAR = 1.2\%$, $FRR = 41\%$) and over the whole ROC curve (5%-$FOM = 66.8\%$).

However, the accuracy curves reveal that the trigram phoneme LM is rather sensitive to changes of the penalty parameters, whereas the exact model behaves more stable with respect to tree node accuracy. Such stability differences between exact and n-gram phoneme LM are generally observed. This plays an important role for practical operation, where a system with stable accuracy may be more desirable than one with better maximum accuracy, but greater sensitivity towards parameter or environment changes.

OOV phoneme LM	xval			eval	
	5%-*FOM*	Acc_n	$(p_{oov}^{in}, \lambda_{oov})$	5%-*FOM*	Acc_n
bigram	58.2%	84.3%	$(-6, 7)$	64.3%	83.6%
trigram + bigram backoff	60.9%	85.0%	$(-3, 7)$	64.0%	85.8%
trigram without backoff	57.1%	85.4%	$(-2, 7)$	**66.8**%	**87.4**%

Table 6.2: *OOV/IV model performance for different n-gram phoneme LM based OOV models, trained on Phonolex, on* 8%*-set.*

As discussed in Section 6.1, open-vocabulary systems with explicit OOV models are expected to avoid the errors that occur in the surrounding of OOV words inevitably misrecognized by closed-vocabulary systems. In order to measure this effect, we use the best closed-vocabulary system of Chapter 5 as basis for comparison of tree node accuracy. It needs to be taken into account, however, that the open-vocabulary systems are trained and evaluated on modified corpus annotations, where semantically irrelevant words have been declared as unknown (compare Section 6.3). Due to the resulting differences in vocabulary size, these systems are strictly not comparable. Therefore, the closed-vocabulary system is rebuilt on the 8% and 23%-sets to obtain baseline systems. The total tree node accuracy Acc_n for these systems is 82.0% and 73.2% on the evaluation set, and 82.9% and 75.2% on the cross-validation set, respectively.

The baseline value for the 8%-set is also shown in Figure 6.5. The exact phoneme LM based open-vocabulary system is able to outperform this baseline over the whole 5%-range of *FAR*. With the trigram phoneme LM, the same is achieved for $FAR \leq 3\%$. At the operating points, the two OOV models achieve relative error rate reductions of $Err_{rel} = -28.9\%$ and $Err_{rel} = -30.0\%$, respectively.

As discussed in Section 6.2, the generalization abilites of n-grams may lead to performance degradations of phoneme LM because of over-generation, especially if backoff to lower-order n-grams is enabled. Therefore, it may be desirable to limit generalization by preventing backoff. In order to examine this issue, three different n-gram phoneme LM were built:

- A bigram LM, where the likelihood of a triphone within an OOV word always depends on the preceding triphone only.
- A trigram LM with bigram backoff, where the bigram likelihood is used if the trigram is missing. This model allows less pronunciations than the bigram LM.
- A trigram LM without backoff to bigrams. In this model, the likelihood of a triphone always depends on the two preceding triphones. This yields the most restrictive pronunciation model.

OOV LM	smoothing	xval			eval	
		5%-*FOM*	Acc_n	$(p_{oov}^{in}, \lambda_{oov})$	5%-*FOM*	Acc_n
exact	add	52.6%	85.8%	$(-3, 5)$	**61.5**%	**87.2**%
	gtnet	52.5%	85.8%	$(-4, 2)$	61.5%	87.1%
trigram	katz	55.3%	85.3%	$(-2, 6)$	64.2%	87.2%
	knmod	57.1%	85.4%	$(-2, 7)$	**66.8**%	**87.4**%

Table 6.3: *System performance for different smoothing techniques on* 8%*-set.*

The unigram backoff is always removed to ensure that the triphones are traversed in the correct order. The performances of these models on the 8%-set are listed in Table 6.2. On the evaluation set, the trigram model without backoff is the clear winner both in terms of OOV detection performance (5%-*FOM* = 66.8%) and regarding semantic accuracy (Acc_n = 87.4%). Hence, this n-gram model variant is used for other experiments. Although the trigram phoneme LM with bigram backoff achieves the best OOV detection performance on the cross-validation set, it cannot confirm this value on the evaluation set. There, it obtains similar detection performance as the bigram model (64.0% vs. 64.3%), but substantially better semantic accuracy (85.8% vs. 83.6%). As expected, less restrictive OOV pronunciation models require higher penalization, because they are more easily confused with IV words.

As for the hierarchical language modeling approach of Chapter 5, two smoothing techniques are investigated for each of the two types of phoneme LM (compare Section 5.3): Katz smoothing (katz) and modified Kneser-Ney smoothing (knmod) for n-gram LM, and additive discounting (add) and Good-Turing discounting (gtnet) for exact LM. The results are shown in Table 6.3. As in Chapter 5, additive discounting and Good-Turing discounting perform similarly. Here, additive discounting performs marginally better. For n-gram phoneme LM, modified Kneser-Ney smoothing again yields consistently better results than Katz smoothing on cross-validation and evaluation sets. Substantial improvements are obtained for the 5%-*FOM* measure (66.8% vs. 64.2% on the evaluation set), whereas the difference in tree node accuracy is rather small (87.4% vs. 87.2%). The best methods, i.e. additive discounting and modified Kneser-Ney smoothing, are again selected as defaults for other experiments.

6.5.3 OOV Training Lexica

So far, experiments were conducted with Phonolex as training lexicon for OOV pronunciation models. In the following, we present comparisons with the second training lexicon, named Celex. As outlined in Section 6.2, the pronunciations of 314k wordforms are used from Celex, whereas Phonolex yields manually verified pronunciations of 22k inflected words. The sizes of these training sets are reduced optionally by selecting only the most frequent pronunciations according to word frequency statistics from Celex (see Section 6.2). For OOV models based on Phonolex,

training lexicon	OOV LM	# pron.	xval			eval	
			5%-*FOM*	Acc_n	$(p_{oov}^{in}, \lambda_{oov})$	5%-*FOM*	Acc_n
Phonolex	exact	22k	52.6%	85.8%	$(-3, 5)$	**61.5**%	**87.2**%
		10k	52.2%	85.8%	$(-3, 5)$	60.1%	86.9%
		1k	45.9%	84.4%	$(-2, 2)$	59.2%	86.7%
	trigram	22k	57.1%	85.4%	$(-2, 7)$	**66.8**%	**87.4**%
		10k	55.8%	85.5%	$(-2, 7)$	63.7%	87.0%
Celex	exact	314k	54.2%	85.2%	$(-3, 6)$	**62.7**%	**87.1**%
		100k	54.1%	85.3%	$(-5, 1)$	61.9%	86.8%
		10k	50.6%	85.7%	$(-2, 6)$	59.8%	87.6%
	trigram	314k	59.4%	84.5%	$(-4, 7)$	**63.2**%	**85.4**%
		100k	57.4%	84.7%	$(-2, 7)$	61.5%	85.8%
closed-vocabulary baseline			n.a.	82.9%	n.a.	n.a.	82.0%

Table 6.4: *Performance for different OOV model training lexica on* 8%*-set.*

reduced lists consisting of 10k and 1k pronunciations are used to build exact and trigram phoneme LM. Since the 1k pronunciations are not sufficient for trigram LM estimation, only exact LM are generated from this list. For the same reason, exact LM are built from 100k and 10k Celex pronunciations, but trigram LM only from the 100k list.

Table 6.4 gives an overview over the detection performance and semantic accuracy of the corresponding OOV word models. The Celex based exact phoneme LM generally display similar performance as the ones based on Phonolex, yet they require a larger number of pronunciations to achieve this. The high semantic accuracy of 87.6% of the exact 10k Celex model can presumably be viewed as an outlier. For trigram phoneme LM, however, the Phonolex pronunciations yield fundamentally better performance (87.4% vs. 85.4% accuracy and 66.8% vs. 63.2% *FOM*). Hence, Phonolex seems to be more suitable for the OOV modeling task considered here. This may be caused by the manual pronunciation verification of Phonolex and by the use of inflected words instead of automatically generated wordforms. Furthermore, the poor match between the phoneme sets of Celex and ODINS certainly plays a role, too.

As expected, reducing the sizes of the OOV training lexica by selecting most frequent words generally causes performance degradations. For all open-vocabulary configurations, however, the maximum accuracy is well above the baseline values of the closed-vocabulary system. The degradations are more consistent for the Phonolex configurations than for those based on Celex. For example, the 100k trigram model achieves $Acc_n = 85.8\%$ on the evaluation set, whereas the 314k model only obtains 85.4%. A reason for this may also be that the pronunciation model becomes so general that its confusability with IV words outweighs the benefits of its larger coverage.

OOV LM	freq. weight.	IV exclusion	xval			eval	
			5%-*FOM*	Acc_n	$(p_{oov}^{in}, \lambda_{oov})$	5%-*FOM*	Acc_n
exact	no	no	52.6%	85.8%	$(-3, 5)$	61.9%	87.0%
	no	yes	51.6%	85.5%	$(-3, 5)$	60.7%	86.3%
	yes	no	52.6%	85.8%	$(-3, 5)$	**61.5**%	**87.2**%
	yes	yes	51.6%	85.5%	$(-4, 1)$	60.4%	86.8%
trigram	no	no	53.2%	85.2%	$(-2, 7)$	66.6%	86.6%
	no	yes	54.0%	85.3%	$(-3, 5)$	65.3%	86.3%
	yes	no	57.1%	85.4%	$(-2, 7)$	**66.8**%	**87.4**%
	yes	yes	56.9%	85.2%	$(-2, 7)$	66.7%	86.6%

Table 6.5: *Influence of frequency weighting and exclusion of IV word pronunciations from OOV training on system performance.*

Furthermore, we test the influence of weighting the pronunciations for OOV model training according to the word frequency information of Celex, and the effects of excluding pronunciations of IV words from the training list (compare Section 6.2). The former aims to improve the frequency distribution of OOV pronunciation models, whereas the latter is expected to lower confusions between OOV and IV words. The experiment, whose results are summarized in Table 6.5, is carried out on exact and trigram phoneme LM for the standard 8%-set Phonolex configuration.

In terms of tree node accuracy Acc_n, the OOV models with word frequency weighting consistently outperform those without on the evaluation set. The same effect can be observed for the 5%-*FOM* of trigram OOV LM on cross-validation and evaluation sets. The *FOM* of exact OOV LM, however, is identical on the cross-validation set and slightly lower on the evaluation set (61.5% vs. 61.9% and 60.4% vs. 60.7%) if word frequency weighting is carried out. Because of the slightly higher semantic accuracy values (87.2% vs. 87.0% and 86.8% vs. 86.3%), the frequency weighted exact models are still used as default for other experiments.

The exclusion of IV word pronunciations consistently degrades both accuracy and *FOM* on the evaluation set, and with one exception also on the cross-validation set. At first, it seems surprising that this measure doesn't succeed in improving system performance. Yet, it can be explained by the fact that some of the known OOV words also appear as IV words within semantic concepts. Therefore, although excluding their pronunciation from the OOV model training set may help to discriminate OOV and IV word models, this also prevents their detection as OOV words. Hence, the desired effect cannot be achieved.

6.5.4 Varying OOV Rates

In the final experiment of this chapter, we examine the robustness of the presented lexical OOV modeling methods against different rates of OOV words. For this pur-

training lexicon	OOV LM	anno. set	xval			eval	
			5%-*FOM*	Acc_n	$(p_{oov}^{in}, \lambda_{oov})$	5%-*FOM*	Acc_n
Phonolex	exact	23%	59.9%	83.7%	$(-2, 7)$	67.3%	**86.3**%
		8%	52.6%	85.8%	$(-3, 5)$	61.5%	87.2%
	trigram	23%	61.7%	83.5%	$(-2, 7)$	**69.1**%	85.6%
		8%	57.1%	85.4%	$(-2, 7)$	**66.8**%	**87.4**%
Celex	exact	23%	59.8%	83.8%	$(-3, 6)$	**68.5**%	**85.8**%
		8%	54.2%	85.2%	$(-3, 6)$	62.7%	**87.1**%
	trigram	23%	60.5%	81.7%	$(-2, 7)$	66.6%	84.3%
		8%	59.4%	84.5%	$(-4, 7)$	**63.2**%	85.4%
closed-vocabulary system		23%	n.a.	75.2%	n.a.	n.a.	73.2%
		8%	n.a.	82.9%	n.a.	n.a.	82.0%

Table 6.6: *Performance of OOV word models for different OOV rates.*

pose, both the 8%-set and the 23%-set are used. For these two annotation sets, OOV detection perfomance and semantic accuracy results for exact and trigram phoneme LM based on the two different training lexica are listed in Table 6.6, along with the baseline values of the corresponding closed-vocabulary systems. The comparison between Phonolex and Celex based OOV models on the 23%-set yields similar results as on the 8%-set: Exact LM perform similarly on both lexica, whereas Phonolex based trigram phoneme LM obtain considerably better performance than those based on Celex (69.1% vs. 66.6% *FOM* and 85.6% vs. 84.3% semantic accuracy).

Again, it is hard to determine a clear winner between exact and trigram OOV LM for all system configurations. Depending on the situation, one or the other model type achieves better evaluation results. Trigrams are e.g. advantageous for Phonolex based models on the 8%-set, whereas exact LM score better for Celex based models on the 23%-set. With other system configurations, 5%-*FOM* and Acc_n even show opposite preferences. However, as outlined earlier in this section, it needs to be taken into account that exact phoneme LM are generally less sensitive towards variations of the OOV model penalization parameters.

Although the performance values of the 8%- and 23%-sets shouldn't be compared directly because of the different system vocabularies, it can nevertheless be noted that OOV detection performance is consistently higher on the 23%-set, whereas the 8%-set shows consistently better semantic accuracy values. In comparison to the closed-vocabulary system, all tested open-vocabulary system configurations obtain substantial gains in semantic accuracy. For the 23%-set, the tree node accuracy of the best open-vocabulary system of $Acc_n = 86.3\%$ corresponds to a relative error rate reduction of $Err_{rel} = -48.9\%$ over the closed-vocabulary baseline ($Acc_n = 73.2\%$).

Chapter 7

Conclusion and Outlook

This work dealt with the task of automatically understanding the meaning of naturally spoken user utterances in limited-domain human-machine communication applications. Practically, this problem was studied by means of a spoken language dialogue application for a German airport information scenario. Most existing speech understanding systems perform similar tasks by loosely coupling automatic speech recognition and natural language understanding in a sequential, two-stage decoding process. For this thesis, a tightly coupled one-stage decoding method was developed, which directly transforms speech into tree-based meaning representations and thereby aims to avoid errors caused by early decisions. In order to lower decoder complexity and maintain the flexibility to integrate new knowledge sources, the one-stage approach was based on a uniform, stochastic knowledge representation that can be processed by a generic decoding algorithm.

The uniform knowledge representation for the airport information test system consists of models for phoneme HMM, word pronunciations, word classes and semantic concepts. Uniformity was realized by explicitly hierarchical modeling through so-called weighted transition network hierarchies (WTNH). Thereby, the different knowledge sources were arranged on 6 hierarchy levels, which reflect the logical processing stages. In order to maintain modeling flexibility, the ability to define sub-levels and to skip certain hierarchy levels was introduced. As an alternative to our explicit approach, an implicit representation of hierarchical structures through weighted finite-state transducers was considered. This technique recently received much attention from speech recognition researchers, mainly because it allows flexible composition and automatic, global optimization of automata. On the downside, hierarchical knowledge needs to be encoded implicitly into WFST, which together with automata optimization complicates tasks that require direct access to the hierarchical model structure, such as confidence measurement or dynamic system reconfiguration. Furthermore, the basic building blocks of WTNH are equivalent to weighted finite-state acceptors, and therefore lend themselves at least to local optimization. WTNH were therefore preferred for the speech interpretation task pursued in this work.

After deriving a generalized formulation of the hierarchical search problem, the basic one-stage decoding scheme was described. It was based on Young's approach to realize a time-synchronous, hierarchical Viterbi search by token passing. As a novelty of our method, the search was directly performed on the explicitly hierarchical WTNH representation, which implies that transition network nodes accommodate multiple tokens at once. This was achieved by augmenting network nodes with token containers, which enables joint propagation of token sets and thereby reduces computational effort in comparison to Young's approach. In contrast to a similar decoding approach, transition networks were processed in topological order if possible. This avoids redundant computations since converging tokens meet as early as possible.

In order to quantify the theoretical advantage of the one-stage decoder resulting from its ability to consider all available knowledge sources simultaneously, a novel experimental comparison with a two-stage decoder was devised. Both systems were based on identical modeling approaches, decoding schemes and training data, so that solely the effect of propagating only a limited number of hypotheses from the first to the second stage could be measured. The experiment showed that the two-stage system increased the error rate on the semantic representation by more than 38% in comparison to the one-stage system if only the best hypothesis is taken into account. As expected, the two-stage system was able to eliminate this difference by use of alternative recognizer hypotheses. However, about 30 alternatives for each word in the lattice were necessary to reduce error rates considerably, and more than 100 alternatives had to be present to achieve the same semantic accuracy as the one-stage system. The resulting computational effort may well exceed the resources available for practical system operation.

Speech data was recorded in a series of Wizard-of-Oz experiments for a German airport information dialogue system scenario. The collected user utterances, which consist of about 15k words in total, were fully annotated with semantic trees. A semi-automatic annotation procedure was developed to reduce manual labor. The resulting corpus was used to train and test WTNH based speech interpretation systems.

For system centric evaluation of our speech interpretation system, standard evaluation measures such as word accuracy, slot-value accuracy and test-set perplexity were considered. Additionally, a novel evaluation measure, the so-called semantic tree node accuracy, was devised to directly assess the semantic tree representation produced by the WTNH based one-stage decoder. In contrast to standard approaches, the novel measure was computed by tree matching instead of sequence matching, and was therefore expected to display greater flexibility when assessing partially correct semantic tree representations. An experimental comparison between sequence matching and tree matching based evaluation measures revealed that this flexibility is especially important when more complex semantic tree structures, as they occur in this work, are evaluated. For this case and if subsequent processing stages are able to capitalize on partly correct semantic representations, the novel evaluation measure was considered more appropriate.

Robust semantic modeling techniques were developed to define the 'upper' part of WTNH, consisting of words, word classes and semantic concepts. Instead of performing explicit syntactic and morphological analyses, robust semantic models are directly based on the word level. As a special characteristic of this work, the proposed hierarchical language models (HLM) were created by combining different rule-based and data-driven language modeling techniques. In particular, weighted context-free rewrite rules, n-gram and exact language models and their representation as weighted transition networks were employed. In order to select the most appropriate model type for each individual HLM part, a decision principle was formulated. Complexity of target language, availability of training data and generalization capabilities of language models were used as decision criteria.

Robust semantic modeling, smoothing of likelihood distributions, combination of data-driven and rule-based language modeling techniques, unknown word modeling and special models for pauses, non-speech sounds and filler words render WTNH suitable to process natural speech. Semantic accuracy values of up to 88% on the evaluation data confirm the effectiveness of the approach.

The influence of smoothing the likelihood distributions of local language models was examined for different techniques. In order to enable data-driven weighting and smoothing of rule-based language models, these operations were directly performed on the network level. By applying modified Kneser-Ney smoothing instead of Katz smoothing, the semantic error rate could be reduced by up to 7%.

Furthermore, the likelihood distribution of HLM was improved by introducing different scaling and penalization parameters. In order to describe the effects of these parameters on all and individual hierarchy levels, appropriate metrics were defined. Experiments showed that within-HLM scaling alone yields error rate reductions of 3% on the concept level and 14% on the word class level. Together with penalization of word insertions, the error rate reductions could be increased to 6% on the concept level.

Speech interpretation systems are not required to produce perfect word transcriptions, since their original goal is the recognition of meaning. Nonetheless, the transcription capabilities of our speech interpretation system and a speech recognition system trained on the same speech corpus were compared. The experiment suggested that the use of semantic knowledge reduces the error rate by 5%.

Consideration of out-of-vocabulary (OOV) words is especially important for speech interpretation systems operating in narrow application domains, if users should be allowed to talk freely. In these circumstances, high rates of OOV words are likely to occur, which are silently misrecognized as In-Vocabulary (IV) words in standard closed-vocabulary systems. In this thesis, explicit OOV modeling was performed with generic, statistical pronunciation models for arbitrary words, since OOV words were expected to be mostly common words from the target language. The pronunciation models were trained on large pronunciation lexica, and optionally weighted according to word frequency statistics. Different aspects of the devised methods are novel, e.g. that OOV models were integrated into WTNH and therefore directly affect the whole speech interpretation process. Furthermore, experiments

were conducted for different OOV rates and with OOV models created by use of different language modeling and smoothing techniques. Evaluation was carried out by creating combined plots of receiver-operating-characteristics and semantic accuracy curves.

Experiments revealed that phoneme language models based on n-grams are rather sensitive to changes of the penalty parameters, whereas those based on exact models behave more stable with respect to semantic accuracy. By weighting the pronunciations used for OOV model training according to word frequency statistics, the semantic error rate could be reduced by 6%. For a test set with a total OOV rate of 8%, 59% of the OOV words could be detected correctly, while only 1% of the IV words were wrongly recognized as OOV words. Simultaneously, the semantic error rate could be reduced by 30% in comparison to the closed-vocabulary system. On a second test set with 23% OOV words, an OOV detection rate of 66% could be achieved for 1% wrongly accepted IV words. For this set, the semantic error could be reduced by up to 49%.

To sum up, it can be said that the presented approaches are well suited for one-stage interpretation of natural speech in limited-domain human-machine communication applications. Yet, some aspects could be investigated more closely. For example, further evaluation measures for semantic tree representations could be formulated, which consider the 'importance' of semantic objects. Thereby, evaluation and also tuning of system parameters could take the characteristics of subsequent modules into account. For instance, semantic categories could be divided into those whose members need to be distinguished for further processing (e.g. airline codes such as AF or BA) and into those whose members are interchangeable in the context of the application (e.g. phrases initiating temporal questions such as 'where' or 'at what time'). Semantic category members of the latter type could then be deleted from semantic trees prior to evaluation. A more elaborate technique could be based on assigning weights to semantic objects corresponding to their importance. These weights could then be considered for computation of semantic accuracy, or directly during semantic tree matching.

Closer consideration of unknown words during semantic processing is another aspect that could be examined further, as most known OOV words appeared between semantic objects in this work. Yet, purposefully integrating OOV word models into semantic categories may further increase speech interpretation robustness. For instance, OOV models could explicitly provide for unknown word class members such as new location names. Thereby, the system would be able to recognize the meaning of larger parts of user utterances, e.g. if the unknown location relates to a flight destination or to a flight origin. It should be noted, however, that such extensions would increase confusability and therefore pose a potential risk to system performance if not applied carefully.

Appendix A

Further Evaluation Results

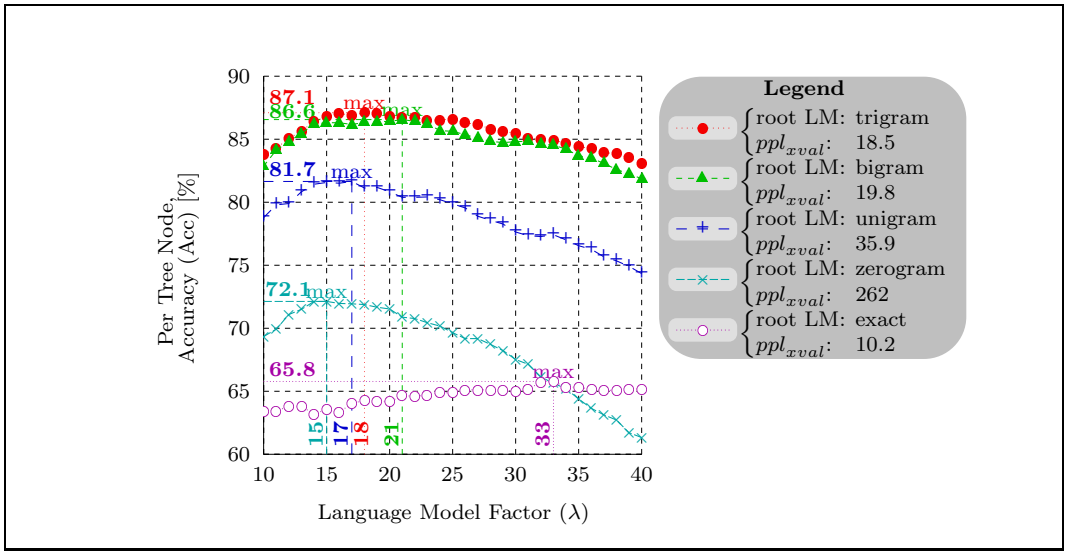


Figure A.1: *Total tree node accuracy for varying n-gram orders of the root LM on the cross-validation set.*

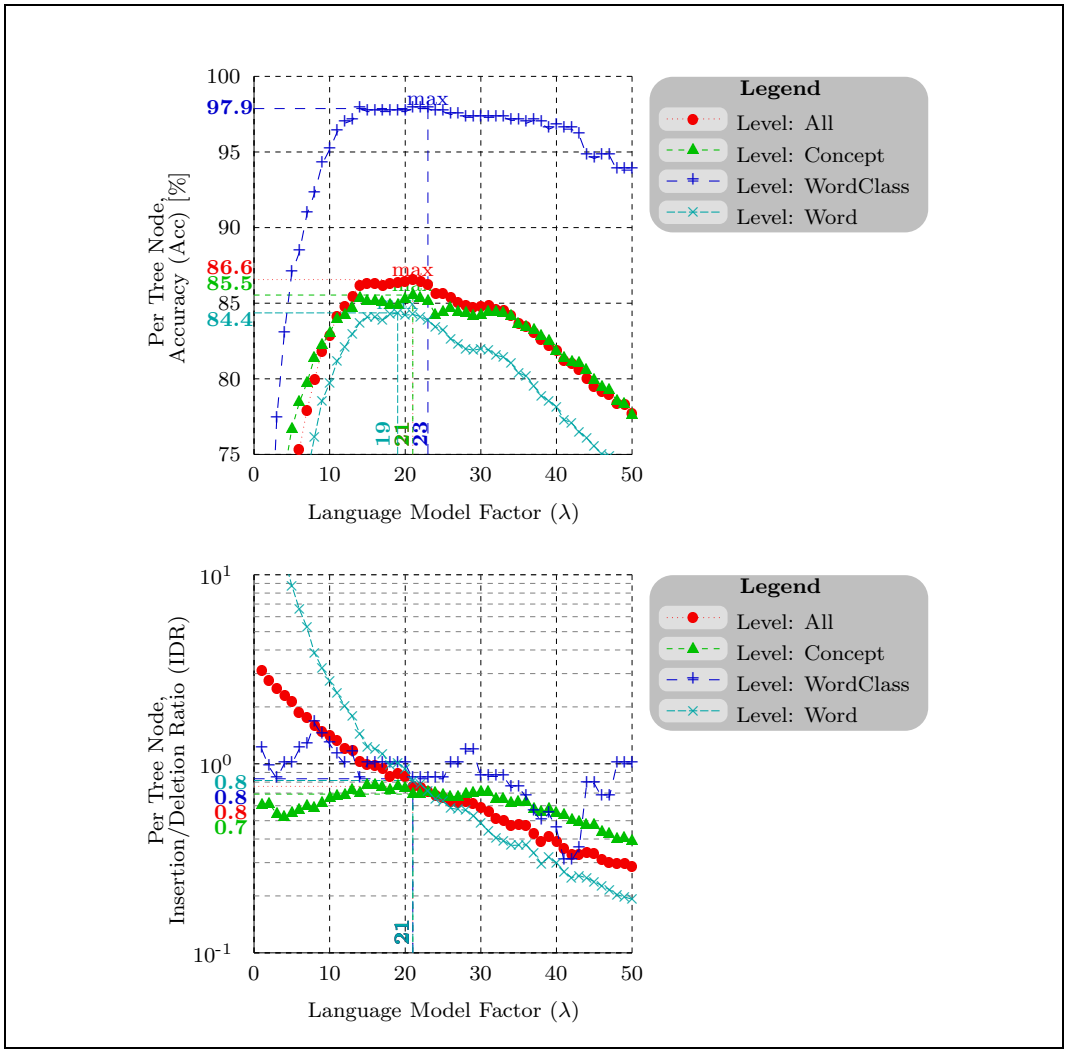


Figure A.2: *Top: Tree node accuracy for all and individual node types for varying λ. Bottom: Insertion-deletion ratio for the identical setup. Both experiments were conducted on the cross-validation set.*

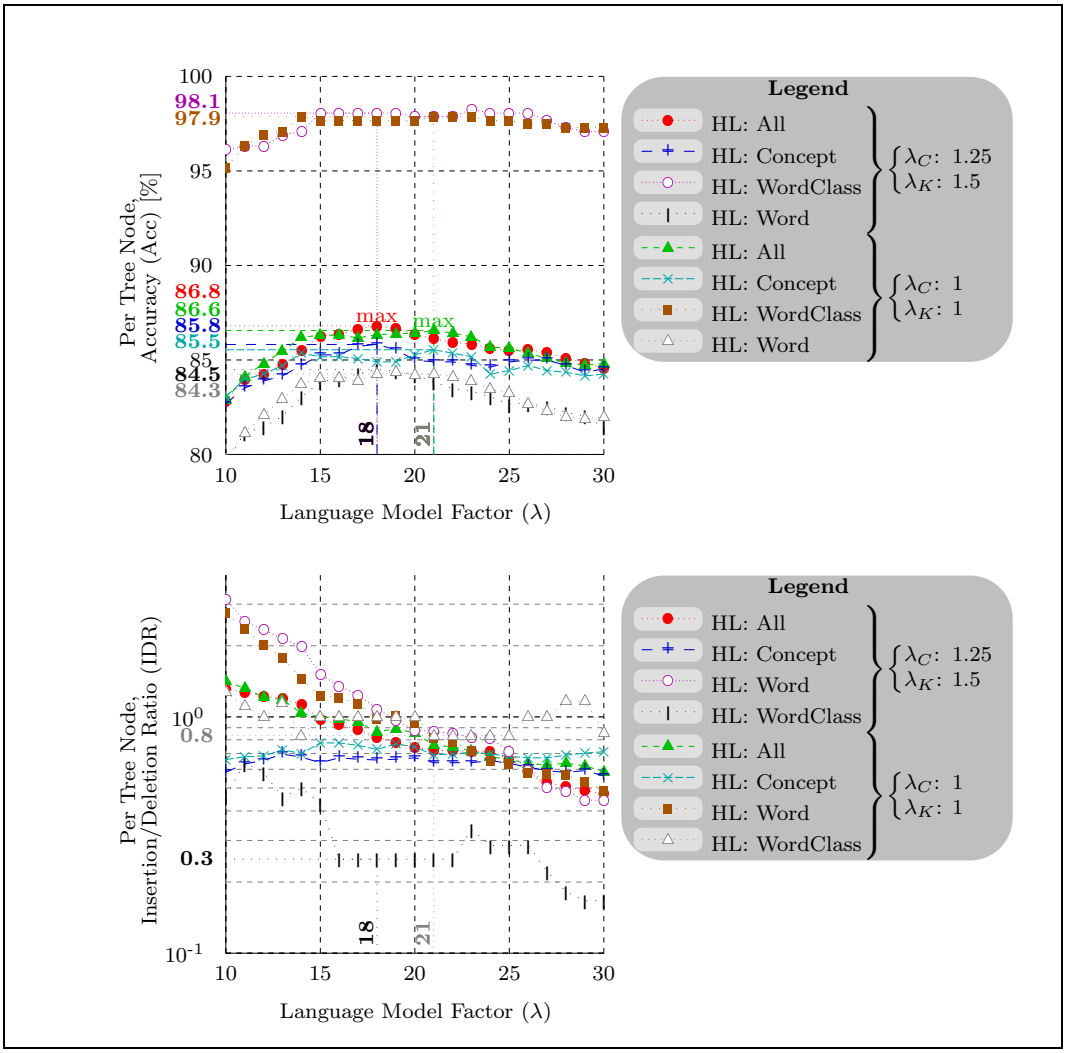


Figure A.3: *Tree node accuracies (top) and IDR curves (bottom) with and without level-dependent scaling factors on the cross-validation set.*

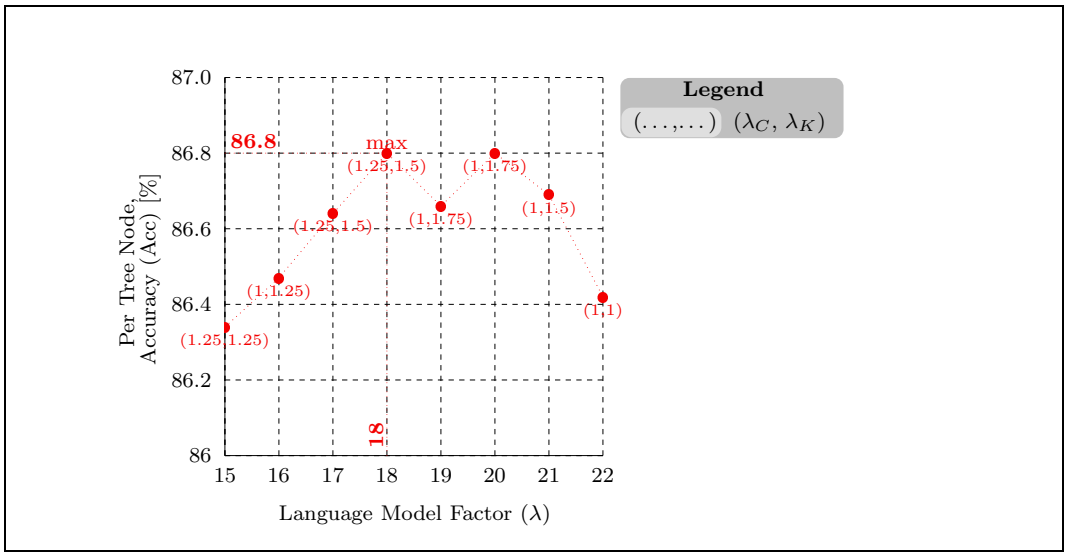


Figure A.4: *Total tree node accuracy over λ for different within-HLM scaling factors (λ_C, λ_K) on the cross-validation set. Only the settings of (λ_C, λ_K) yielding maximum accuracy are shown in brackets below the data points. The best accuracy is achieved at $\lambda = 18$, $\lambda_C = 1.25$ and $\lambda_K = 1.5$.*

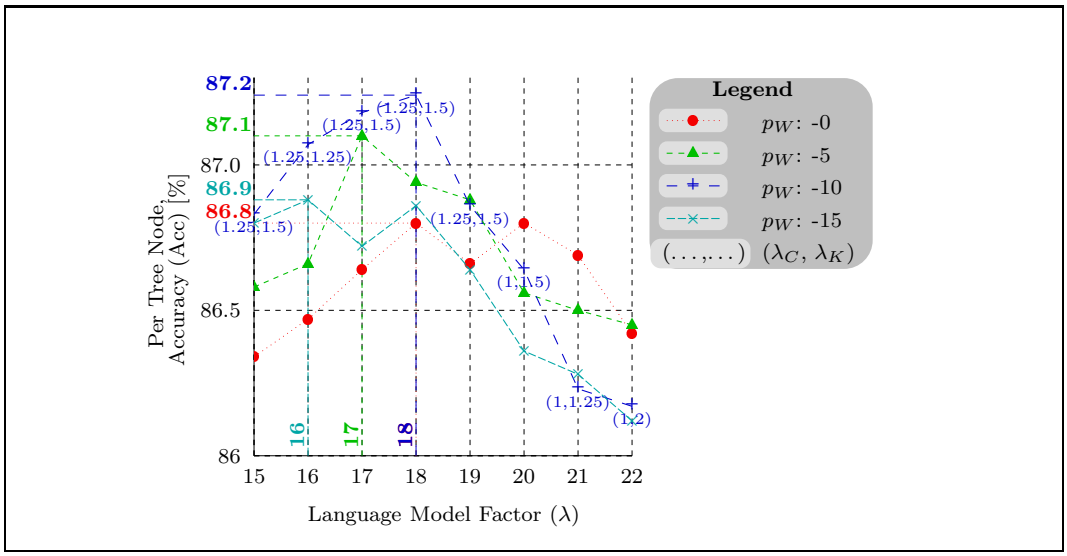


Figure A.5: *Total tree node accuracy over λ for different (λ_C, λ_K) and different word insertion penalties p_W on the cross-validation set. Only the settings of (λ_C, λ_K) yielding maximum accuracy are shown. The best accuracy is achieved at $\lambda = 18$, $\lambda_C = 1.25$, $\lambda_K = 1.5$ and $p_W = -10$.*

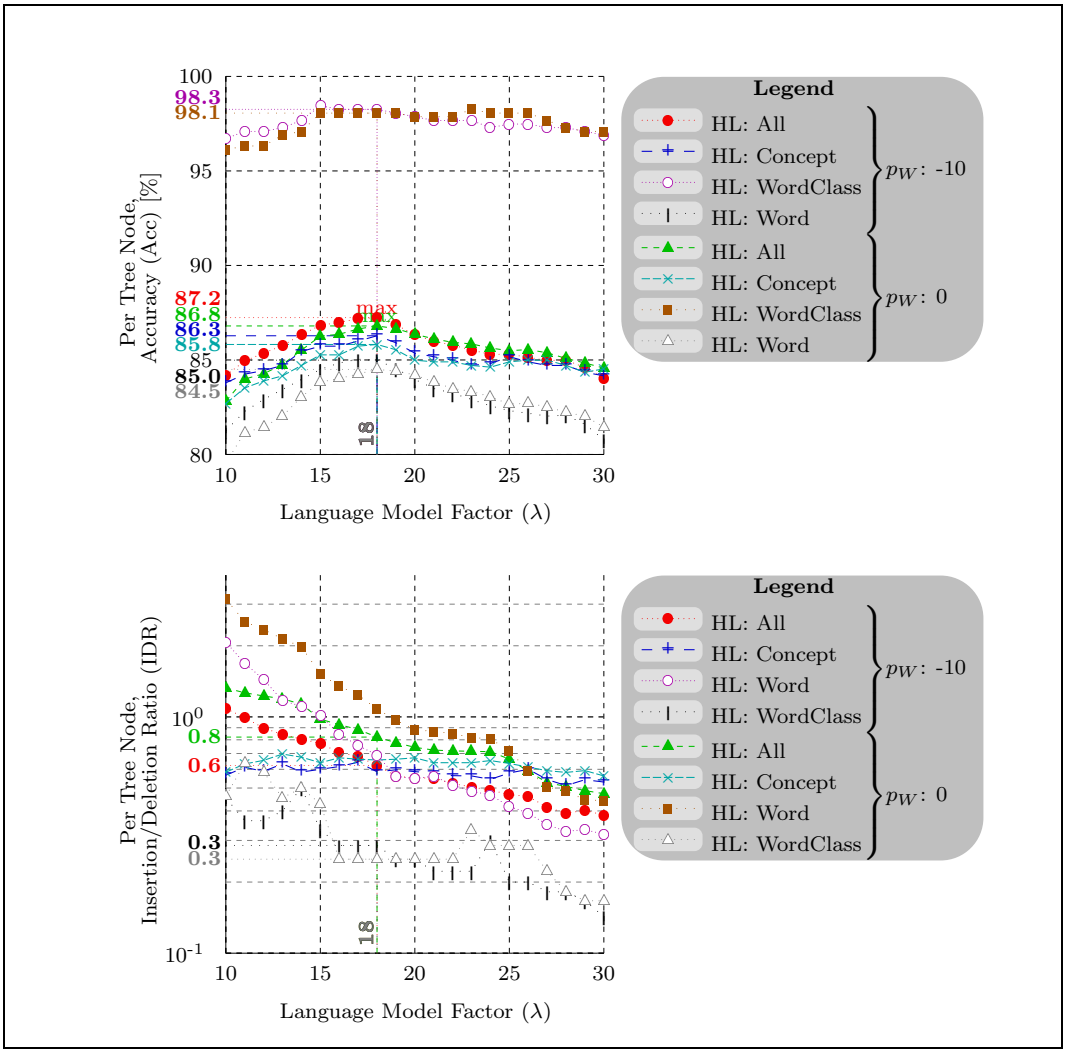


Figure A.6: *Tree node accuracies (top) and IDR curves (bottom) with and without word insertion penalty on the cross-validation set.*

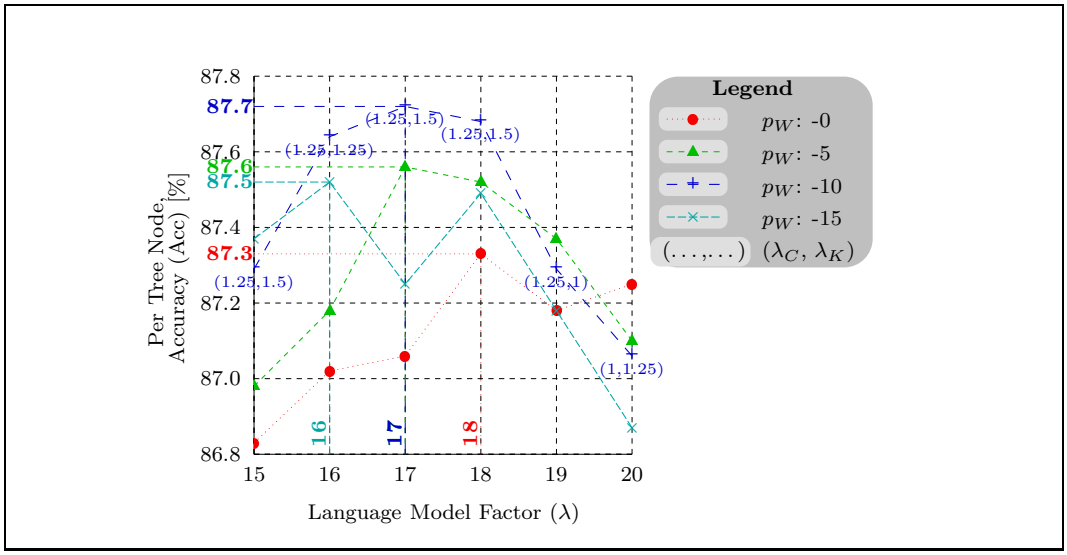


Figure A.7: *Total tree node accuracy (after omitting the concept level) over λ for different p_W and (λ_C, λ_K) for concept based HLM on the cross-validation set. Only the settings of (λ_C, λ_K) yielding maximum joint word and word class accuracy are shown. The best accuracy is achieved at $\lambda = 17$, $\lambda_C = 1.25$, $\lambda_K = 1.5$ and $p_W = -10$.*

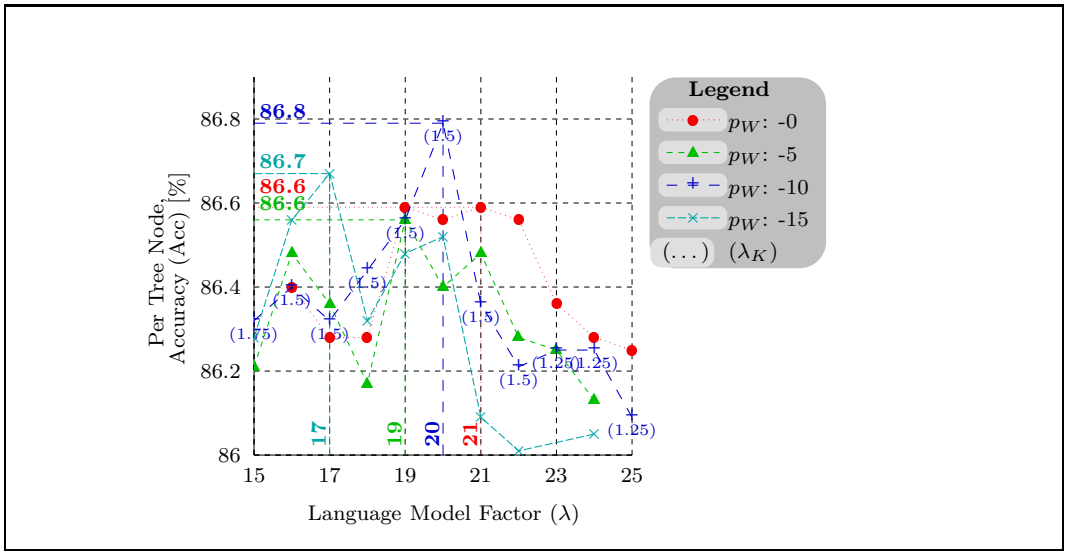


Figure A.8: *Total tree node accuracy over λ for different p_W and λ_K for word class based HLM on the cross-validation set. Only the settings of (λ_K) yielding maximum accuracy are shown. The best accuracy is achieved at $\lambda = 20$, $\lambda_K = 1.5$ and $p_W = -10$.*

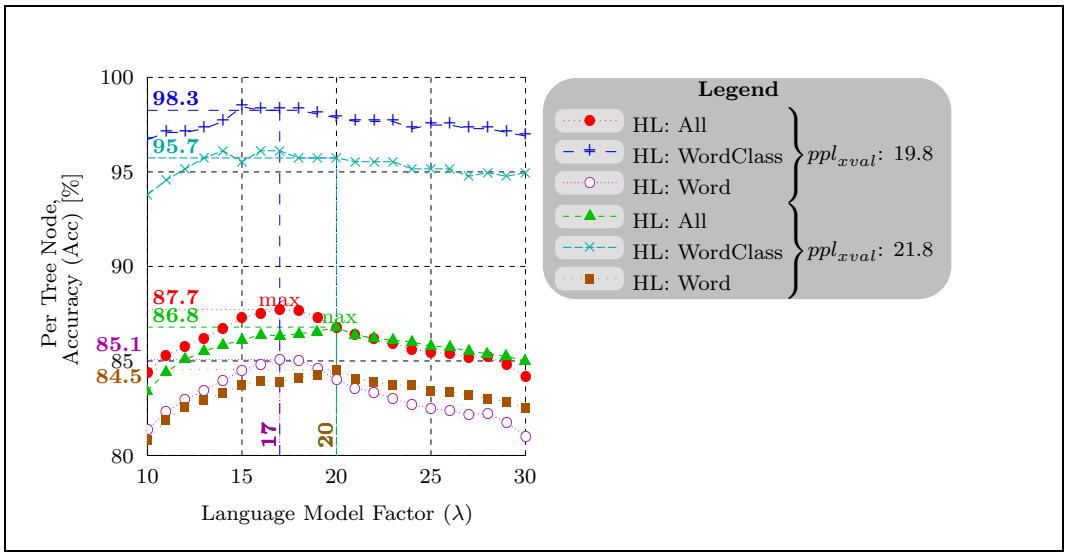


Figure A.9: *Speech interpretation (concept based HLM, upper* 3 *curves of legend) vs. speech recognition (word class based HLM, lower* 3 *curves) on the cross-validation set.*

Bibliography

[AMR03] C. Allauzen, M. Mohri, and B. Roark. Generalized Algorithms for Constructing Statistical Language Models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47, Sapporo, Japan, 2003.

[AW04] A. Acero and Y. Wang. A Semantically Structured Language Model. In *Special Workshop in Maui (SWIM)*, Hawaii, 2004.

[Bak75] J. Baker. The DRAGON System – An Overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, 23:24–29, 1975.

[Baz02] I. Bazzi. *Modelling Out-of-Vocabulary Words for Robust Speech Recognition.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, Massachusetts, 2002.

[BB75] J. Brown and R. Burton. Multiple Representations of Knowledge for Tutorial Reasoning. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding*, pages 311–349. Academic Press, New York, 1975.

[BD00] N. Bernsen and L. Dybkjaer. Is that a Good Spoken Language Dialogue System? In *Proceedings of the LREC'2000 Satellite Workshop 'Using Evaluation within HLT Programs: Results and Trends'*, pages 47–49, Athens, Greece, 2000.

[BDHS95] H. Bratt, D. Dowding, and K. Hunicke-Smith. The SRI Telephone-based ATIS System. In *Proceedings of the Spoken Language Systems Technology Workshop*, Austin, Texas, 1995.

[BEG+96] M. Boros, W. Eckert, F. Gallwitz, G. Görz, G. Hanrieder, and H. Niemann. Towards Understanding Spontaneous Speech: Word Accuracy vs. Concept Accuracy. In *Proceedings of the 4th International Conference on Spoken Language Processing (ICSLP)*, volume 2, pages 1009–1012, Philadelphia, PA, 1996.

[BGH96] C. Benoit, M. Grice, and V. Hazan. The SUS test: A method for the assessment of text-to-speech synthesis intelligibility using semantically unpredictable sentences. *Speech Communication*, 18:381–392, 1996.

[BJ04] S. Bangalore and M. Johnston. Balancing Data-driven and Rule-based Approaches in the Context of a Multimodal Conversational System. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, pages 33–40, Boston, Massachusetts, 2004.

[Bod98] R. Bod. Spoken Dialogue Interpretation with the DOP Model. In C. Boitet and P. Whitelock, editors, *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and Seventeenth International Conference on Computational Linguistics (COLING/ACL)*, pages 138–144, San Francisco, California, 1998. Morgan Kaufmann Publishers.

[BPG95] R. Baayen, R. Piepenbrock, and L. Gulikers. The CELEX Lexical Database (CD-ROM). Linguistic Data Consortium, University of Pennsylvania [Distributor], 1995.

[Car95] G. Carroll. *Learning Probabilistic Grammars for Language Modelling*. PhD thesis, Department of Computer Science, Brown University, 1995.

[CBR98] S. Chen, D. Beeferman, and R. Rosenfeld. Evaluation Metrics for Language Models. In *DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[CG98] S. Chen and J. Goodman. An Empirical Study of Smooting Techniques for Language Modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University, Cambridge, Massachusetts, 1998.

[Che96] S. Chen. *Building Probabilistic Models for Natural Language*. PhD thesis, Harvard University, Cambridge, Massachusetts, 1996.

[Cho56] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

[DGP99] N. Deshmukh, A. Ganapathiraju, and J. Picone. Hierarchical Search for Large Vocabulary Conversational Speech Recognition. *IEEE Signal Processing Magazine*, 16(5):84–107, 1999.

[DJA93] N. Dahlbäck, A. Jönsson, and L. Ahrenberg. Wizard of Oz Studies – Why and How. In *Workshop on Intelligent User Interfaces*, Orlando, Florida, 1993.

[FAM96] G. Ferguson, J. Allen, and B. Miller. TRAINS-95: Towards a Mixed-Initiative Planning Assistant. In *Artificial Intelligence Planning Systems*, pages 70–77, 1996.

[Fet98] P. Fetter. Detection and Transcription of OOV Words. Technical Report 231, Verbmobil, 1998.

[FF93] W. Fisher and J. Fiscus. Better Alignment Procedures for Speech Recognition Evaluation. In *Proceedings of the 18th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 59–62, Minneapolis, Minnesota, 1993.

[FLRT05] T. Fabian, R. Lieb, G. Ruske, and M. Thomae. A Confidence-Guided Dynamic Pruning Approach – Utilization of Confidence Measurement in Speech Recognition –. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech)*, Lisbon, Portugal, 2005.

[GA00] L. Galescu and J. Allen. Hierarchical Statistical Language Models: Experiments on In-Domain Adaptation. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, pages 186–189, Beijing, China, 2000.

[GAB+98] F. Gallwitz, M. Aretoulaki, M. Boros, J. Haas, S. Harbeck, R. Huber, H. Niemann, and E. Nöth. The Erlangen Spoken Dialogue System EVAR: A State-of-the-Art Information Retrieval System. In *Proceedings of 1998 International Symposium on Spoken Dialogue (ISSD 98)*, pages 19–26, Sydney, Australia, 1998.

[GC94] W. Gale and K. Church. What's Wrong with Adding One? In N. Oostdijk and P. de Haan, editors, *Corpus-Based Research into Language: In Honour of Jan Aarts*, pages 189–200. Rodopi, Amsterdam, 1994.

[Goo53] I. Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 40:237–264, 1953.

[HC96] Y. Han and K.-S. Choi. A Chart Re-estimation Algorithm for a Probabilistic Recursive Transition Network. *Computational Linguistics*, 22(3):421–429, 1996.

[HU79] J. Hopcroft and J. Ullman, editors. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.

[HY03] Y. He and S. Young. A Data-Driven Spoken Language Understanding System. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, St. Thomas, U.S. Virgin Islands, 2003.

[Jel76] F. Jelinek. Continuous Speech Recognition by Statistical Methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

[JHHS00] M. Johnsen, T. Holter, E. Harborg, and T. Svendsen. Stochastic modeling of semantic content for use in a spoken dialogue system. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, pages 218–221, Beijing, China, 2000. ISCA.

[JM00] D. Jurafsky and J. Martin, editors. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, New Jersey, 2000.

[JMBB77] F. Jelinek, R. Mercer, L. Bahl, and J. Baker. Perplexity – A Measure of Difficulty of Speech Recognition Tasks. In *Proceedings of the 94th Meeting of the Acoustical Society of America*, Miami Beach, Florida, 1977.

[JZP+01] B. Jelinek, F. Zheng, N. Parihar, J. Hamaker, and J. Picone. Generalized Hierarchical Search in the ISIP ASR System. In *Proceedings of the Thirty-Fifth Asilomar Conference on Signals, Systems, and Computers*, volume 2, pages 1553–1556, Pacific Grove, California, 2001. IEEE.

[Kat87] S. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, 35(3):400–401, 1987.

[KN95] R. Kneser and H. Ney. Improved Backing-off for M-Gram Language Modeling. In *Proceedings of the 20th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 181–184, Detroit, Michigan, 1995.

[LFRT04] R. Lieb, T. Fabian, G. Ruske, and M. Thomae. Estimation of Semantic Confidences on Lattice Hierarchies. In *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP)*, pages 569–572, Jeju Island, Korea, 2004.

[Lid20] G. Lidstone. Note on the General Case of the Bayes-Laplace Formula for Inductive or A Posteriori Probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.

[LR03] R. Lieb and G. Ruske. Natürlich-sprachliche Dialogführung für die Nutzung komplexer Informationsdienste im Automobil (NaDia). Technical report, Lehrstuhl für Mensch-Maschine Kommunikation, Technische Universität München, Munich, Germany, 2003.

[LRGB99] L. Lamel, S. Rosset, J. Gauvain, and S. Bennacef. The LIMSI ARISE System for Train Travel Information. In *Proceedings of the 24th International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 501–504, Phoenix, Arizona, 1999.

[LY90] K. Lari and S. Young. The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[Min98] W. Minker. Evaluation Methodologies for Interactive Speech Systems. In *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC 1998)*, pages 199–206, Granada, Spain, 1998.

[Moh97] M. Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2), 1997.

[MPR98] M. Mohri, F. Pereira, and M. Riley. A Rational Design for a Weighted Finite-State Transducer Library. In Derick Wood and Sheng Yu, editors, *Automata Implementation, Second International Workshop on Implementing Automata (WIA '97)*, volume 1436 of *Lecture Notes in Computer Science*, pages 144–158, Berlin, 1998. Springer-Verlag.

[MPR02] M. Mohri, F. Pereira, and M. Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language*, 16(1):69–88, 2002.

[MPR03] M. Mohri, F. Pereira, and M. Riley. AT&T FSM Library — Finite-State Machine Library. AT&T Labs Research, 2003. Available: `http://www.research.att.com/sw/tools/fsm/`.

[NEK94] H. Ney, U. Essen, and R. Kneser. On Structuring Probabilistic Dependencies in Stochastic Language Modeling. *Computer Speech and Language*, 8:1–38, 1994.

[ON95] S. Ortmanns and H. Ney. Experimental Analysis of the Search Space for 20000-Word Speech Recognition. In *Proceedings of the 4th European Conference on Speech Communication and Technology (Eurospeech)*, pages 901–904, Madrid, Spain, 1995.

[Pec93] J. Peckham. A New Generation of Spoken Dialogue Systems: Results and Lessons from the SUNDIAL Project. In *Proceedings of the 3rd European Conference on Speech Communication and Technology (Eurospeech)*, pages 33–40, Berlin, Germany, 1993.

[pho04] Pronunciation Lexicon PHONOLEX. Bavarian Archive for Speech Signals, 2004. Version 3.11.

[PLV93] R. Pieraccini, E. Levin, and E. Vidal. Learning how to Understand Language. In *Proceedings of the 3rd European Conference on Speech Communication and Technology (Eurospeech)*, pages 1407–1412, Berlin, Germany, 1993.

[Rab89] L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77 2:257–285, 1989.

[RH03] M. Rayner and B. Hockey. Transparent Combination of Rule-based and Data-driven Approaches in Speech Understanding. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 299–306, Budapest, Hungary, 2003.

[RJ86] L. Rabiner and B. Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, pages 4–16, 1986.

[RN95] S. Russel and P. Norvig, editors. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[RPB96] G. Riccardi, R. Pieraccini, and E. Bocchieri. Stochastic Automata for Language Modeling. *Computer Speech and Language*, 10(4):265–293, 1996.

[RPC95] M. Riley, F. Pereira, and P-Y. Chung. Lazy transducer composition: A flexible method for on-the-fly Expansion of context-dependent grammar networks. In *1995 Workshop on Automatic Speech Recognition, IEEE Signal Proc. Society*, 1995.

[Rus94] G. Ruske, editor. *Automatische Spracherkennung: Methoden der Klassifikation und Merkmalsextraktion*. Oldenbourg-Verlag, Munich, Germany, 2nd, extended edition, 1994.

[SF93] A. Simpson and N. Fraser. Black Box and Glass Box Evaluation of the SUNDIAL System. In *Proceedings of the 3rd European Conference on Speech Communication and Technology (Eurospeech)*, volume 2, pages 1423–1426, Berlin, Germany, 1993.

[Sha48] C. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[Six03] A. Sixtus. *Across-Word Phoneme Models for Large Vocabulary Continuous Speech Recognition*. PhD thesis, RWTH Aachen, Chair of Computer Science VI, Aachen, Germany, 2003.

[SK83] D. Sankoff and J. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison Wesley, 1983.

[SML97] H. Stahl, J. Müller, and M. Lang. Controlling Limited-Domain Applications by Probabilistic Semantic Decoding of Natural Speech. In *Proceedings of the 22nd International Conference on Acoustics, Speech,*

and Signal Processing (ICASSP), pages 1163–1166, Munich, Germany, 1997.

[Spr99a] R. Sproat. *Lextools Manual Page, Section 1.* AT&T Labs Research, 1999. Available: `http://www.clsp.jhu.edu/ws99/projects/normal/lextools/lextools.1.html`.

[Spr99b] R. Sproat. *Lextools Manual Page, Section 5.* AT&T Labs Research, 1999. Available: `http://www.clsp.jhu.edu/ws99/projects/normal/lextools/lextools.5.html`.

[Spr03] R. Sproat. Lextools: a toolkit for finite-state linguistic analysis. AT&T Labs Research, 2003. Available: `http://www.research.att.com/sw/tools/lextools/`.

[Sto02] A. Stolcke. SRILM - An Extensible Language Modeling Toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP)*, Denver, Colorado, USA, 2002. ISCA.

[SZ90] B. Shapiro and K. Zhang. Comparing Multiple RNA Secondary Structures using Tree Comparisons. *Computer Applications in the Biosciences (CABIOS)*, 6(4):309–318, 1990.

[SZ97] D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In A. Apostolico and Z. Galil, editors, *Pattern Matching Algorithms*, chapter 14. Oxford University Press, 1997.

[TFLR03a] M. Thomae, T. Fabian, R. Lieb, and G. Ruske. A One-Stage Decoder for Interpretation of Natural Speech. In *Proceedings of the IEEE International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE)*, pages 56–64, Beijing, China, 2003.

[TFLR03b] M. Thomae, T. Fabian, R. Lieb, and G. Ruske. Tree Matching for Evaluation of Speech Interpretation Systems. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 477–482, St. Thomas, U.S. Virgin Islands, 2003.

[TFLR05a] M. Thomae, T. Fabian, R. Lieb, and G. Ruske. Hierarchical Language Models for One-Stage Speech Interpretation. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech)*, Lisbon, Portugal, 2005.

[TFLR05b] M. Thomae, T. Fabian, R. Lieb, and G. Ruske. Lexical Out-of-Vocabulary Models for One-Stage Speech Interpretation. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech)*, Lisbon, Portugal, 2005.

[vBKN99] G. van Noord, G. Bouma, R. Koeling, and M.-J. Nederhof. Robust Grammatical Analysis for Spoken Dialogue Systems. *Journal of Natural Language Engineering*, 5(1):45–93, 1999.

[Wah00] W. Wahlster, editor. *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer, Berlin, Germany, 2000.

[YEK+02] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland. *The HTK Book (for HTK Version 3.2)*. Cambridge University Engineering Department, 2002.

[YRT89] S. Young, N. Russell, and J. Thornton. Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems. Technical Report CUED/F–INFENG/TR.38, Cambridge University Engineering Department, 1989.

[ZSG+00] V. Zue, S. Seneff, J. Glass, J. Polifroni, C. Pao, T. Hazen, and L. Hetherington. JUPITER: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1), 2000.

Index